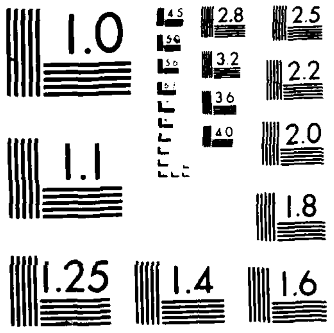
AD-A175 249   A DEDUCTIVE APPROACH TO COMPUTER PROGRAMMING(U)   1/2
STANFORD UNIV CA DEPT OF COMPUTER SCIENCE Z MANNA 1986
AFOSR-TR-86-2164 AFOSR-81-0014

UNCLASSIFIED   F/G 9/2   NL



MICROCOPY RESOLUTION TEST CHART

2

SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 86-2164	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (*and Subtitle*) A DEDUCTIVE APPROACH TO COMPUTER PROGRAMMING		5. TYPE OF REPORT & PERIOD COVERED Final Scientific Report 10/1/84 - 9/30/85
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(*s*) Prof. Zohar Manna		8. CONTRACT OR GRANT NUMBER(*s*) AFOSR 81-0014
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Computer Science Stanford University Stanford, CA 94305		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS United States Air Force Air Force Office of Scientific Research Bldg. 410,Bolling Air Force Base, Wash. DC 20332 nm		12. REPORT DATE
		13. NUMBER OF PAGES 5
14. MONITORING AGENCY NAME & ADDRESS(*if different from Controlling Office*) Same as 11		15. SECURITY CLASS. (*of this report*) unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (*of this Report*)

Approved for public release;
distribution unlimited.

17. DISTRIBUTION STATEMENT (*of the abstract entered in Block 20, if different from Report*)

Approved for public release;
distribution unlimited.

18. SUPPLEMENTARY NOTES

DEC 1 [illegible] 1986

19. KEY WORDS (*Continue on reverse side if necessary and identify by block number*)

20. ABSTRACT (*Continue on reverse side if necessary and identify by block number*)

094120-SH

A DEDUCTIVE APPROACH TO COMPUTER PROGRAMMING

by

Zohar Manna
Professor of Computer Science
Stanford University Stanford, CA 94305

Final Scientific Report:
Air Force Office of Scientific Research
Grant AFOSR-81-0014
Oct. 1, 1984 - Sept. 30, 1985

Summary

Our research was concentrated on the following topics:

- Special Relations in Automated Deduction (Manna and Waldinger [85a][85b])

Theorem provers have exhibited super-human abilities in limited, obscure subject domains but seem least competent in areas in which human intuition is best developed. One reason for this is that an axiomatic formalization requires us to state explicitly facts that a person dealing in a familiar subject would *consider too obvious to mention*; the proof must take each of these facts into account explicitly. A person who is easily able to construct an argument informally may be too swamped in detail to understand, let alone produce, the corresponding formal proof. A continuing effort in our research is to make formal theorem proving more closely resemble intuitive reasoning. One case in point is our treatment of special relations.

In most proofs of interest for program synthesis, certain mathematical relations, such as equality and the orderings, present special difficulties. These relations occur frequently in specifications and in derivation proofs. If their properties are represented axiomatically, proofs become lengthy, difficult to understand, and even more difficult to produce or discover automatically. Axioms such as transitivity have many consequences, most of which are irrelevant to the proof; including them produces an explosion in the search space.

For the equality relation, the approach that was adopted early on is to represent its properties with rules of inference rather than axioms. In resolution systems, two rules of inference, paramodulation (Wos and Robinson [69]) and E-resolution (Morris [69]), were introduced. Proofs using these rules are shorter and clearer, because one application of a rule can replace the application of several axioms. More importantly, we may drop the equality axioms from the clause set, thus eliminating their numerous consequences from the search space.

We have discovered two rules of inference that play a role for an arbitrary relation analogous to that played by paramodulation and E-resolution for the equality relation. These rules apply to sentences employing a full set of logical connectives; they need not be in the clause form required by traditional resolution theorem provers. We intend both these rules to be incorporated into theorem provers for program synthesis.

Employing the new special-relations rules yields the same benefits for an *arbitrary* relation as using paramodulation and E-resolution yields for equality: proofs become shorter and more comprehensible and the search space becomes sparser.

- Binary-Search Algorithms (Manna and Waldinger [85c])

Some of the most efficient numerical algorithms rely on a strategy of *binary search*; according to this strategy, the interval in which the desired output is sought is divided roughly in half at each iteration. This technique is so useful that some authors (e.g., Dershowitz and Manna [77] and Smith [85]) have proposed that a general binary-search paradigm or schema be built into program synthesis systems and then specialized as required for particular applications.

It is certainly valuable to store such schemata if they are of general application and difficult to discover. This approach, however, leaves open the question of how schemata are discovered in the first place. We have found that the concept of binary search appears quite naturally and easily in the derivations of some numerical programs. The concept arises as the result of a single resolution step, between a goal and itself.

The programs we have produced in this way (e.g., real-number quotient and square root, integer quotient and square root, and array searching) are quite simple and reasonably efficient but are bizarre in appearance and different from what we would have constructed by informal means. For example, we have developed the real-number square-root program $sqrt(r, \mathcal{E})$ given above. This program tests if the error tolerance $\mathcal{E}$ is sufficiently large; if so, 0 is a close enough approximation. Otherwise, the program finds recursively an approximation within $2\mathcal{E}$ less than the exact square root. It then tries to refine this estimate, increasing it by $\mathcal{E}$ if the exact square root is large enough and leaving it the same otherwise.

This program was surprising to us in that it doubles a number rather than halving it as the classical binary-search program does. Nevertheless, if the repeated occurrences of the recursive call $sqrt(r, 2\mathcal{E})$ are combined by common-subexpression elimination, this program is as efficient as the familiar one and somewhat simpler.

- Plan Formation in Situational Logic (Manna and Waldinger [85d])

The deductive-tableau approach applies directly to the synthesis of *applicative* (or *functional*) programs, which alter no data structures and produce no other side effects. To apply the same approach to *nonapplicative* programs, which may produce side effects, we have employed a *situational logic*, i.e., a system that allows us to refer explicitly to the states of a computation.

The situational logic we have developed (Manna and Waldinger [81]) fits well within the deductive-tableau framework. We include new functions, such as $val(s, e)$ (the value of expression e in state s), $s;e$ (the state produced by evaluating expression e in state s), and new relations, such as $holds(s, p)$ (true if the value of sentence p is true in state s, and false otherwise). These are ordinary function and relation symbols; proofs in situational logic may employ the ordinary deductive-tableau inference rules.

We are currently attempting to apply these techniques to problems in robot planning by proving theorems in a new formulation of situational logic. Our machine-oriented deductive-tableau inference system is adapted to this logic, with special attention being paid to the derivation of conditionals and recursive plans. With an implementation of the Fay's [79] unification algorithm (see also Hullot [80]), it has been found possible to build in equations and equivalences of the situational logic. Inductive proofs of theorems for even the simplest planning problems have been found to require challenging generalizations.

- Synthesis of Concurrent Programs (Bengt, Manna, Waldinger [86])

The synthesis of concurrent programs is much more complicated than the synthesis of sequential programs. In general, a concurrent program does not have a single input value and a single output value, but receives several inputs and sends several outputs during its execution. If we consider *sequences* of input and output values, then we can specify a concurrent program by giving a relation between the sequence of input values and the sequence of output values. This specification method is natural especially for networks of deterministic processes that communicate asynchronously by sending messages over buffered channels (see e.g. [Kahn 74]). Deterministic data flow networks fall into this category.

We developed a framework for deductive synthesis of such concurrent programs. Since we wanted to use induction conveniently, we considered only networks that produce finite sequences of output values when receiving finite sequences of output values.

- Nonclausal Logic Programming (Malachi, Manna and Waldinger [84] [85], Malachi [86])

A deductive-tableau theorem prover can be adapted to serve as the interpreter for a programming language just as resolution theorem provers have been adapted to interpret the language PROLOG. The programming language TABLOG we obtain in this way combines attractive features of LISP and PROLOG:

- It allows the use of equality in programs. (This is allowed in LISP but forbidden in PROLOG.)
- Programs may define either functions or relations. (LISP programs must represent relations as truth-valued functions; PROLOG programs must represent n-ary functions as $(n+1)$-ary relations.)
- Pattern matching and backtracking are built in. (They are not part of LISP.)

In contrast to other languages combining PROLOG and LISP features, such as LOGLISP (Robinson and Sibert [82]) and QLOG (Komorowski [79]), TABLOG is a single new language, not a meld of two separate components.

A sample TABLOG program, to insert a number u in its place in an ordered list of numbers, is as follows:

$$\begin{array}{lll} insert(u,[\,]) & = & [u] \\ insert(u, v \circ x) & = & if\ \ u \leq v \\ & & then\ \ u \circ (v \circ x) \\ & & else\ \ v \circ insert(u, x) \end{array}$$

Here [ ] is the empty list, $[u]$ is the singleton list whose sole element is u, and $v \circ x$ is the result of inserting ("consing") the number v at the beginning of the list x. We find this program to be clearer than the corresponding program in either LISP or PROLOG.

- Logic: The Calculus of Computer Science

The research papers in which we have presented the deductive approach to program synthesis has been addressed to the customary advanced readership of the scholarly journals. In an effort to make this work accessible to a wider audience, including computer science undergraduates and programmers, we have developed a more elementary treatment in the form of a two-volume book, *The Logical Basis for Computer Programming*, Addison-Wesley (Manna and Waldinger [85]).

The book requires no computer programming and no mathematics other than an intuitive understanding of sets, relations, functions, and numbers; the level of exposition is elementary. Nevertheless, the text presents some novel research results, including

- theories of strings, trees, lists, and finite sets and bags, particularly well suited to theorem-proving and program-synthesis applications;
- formalizations of parsing, infinite sequences, expressions, substitutions, and unification;
- a nonclausal version of skolemization;
- a treatment of stepwise induction in the deductive-tableau framework.

7. Publications

Malachi, Y. [86]

Nonclausal logic programming, Ph.D. thesis (supervised by Z. Manna), Computer Science Department, Stanford University, Stanford, CA, *1986*.

Malachi, Y., Z. Manna, and R. Waldinger [84]

TABLOG: The deductive-tableau programming language, *ACM Symposium on* LISP *and Functional Programming*, Austin, TX, August 1984, pp. 323-330.

Malachi, Y., Z. Manna, and R. Waldinger [85]

TABLOG: Functional and relational programming in one framework, IEEE *software*, Vol. 2, No. 1 (January 1986), pp. 75-76 (invited abstract).

Manna, Z., and R. Waldinger [80]

A deductive approach to program synthesis, *ACM Transactions on Programming Languages and Systems*, Vol. 2, No. 1, January 1980, pp. 90-121.

Manna, Z., and R. Waldinger [81]

Problematic features of programming languages: a situational-calculus approach, *Acta Informatica*, Vol. 16, 1981, pp. 371-426.

Manna, Z., and R. Waldinger [85a]

Special relations in automated deduction, *Journal of the ACM*, Vol. 33, No. 1 (Jan. 1986), pp. 1-60. An abbreviated version appears in the Proceedings of the *Twelfth International Colloquium on Automata, Languages, and Programming* (ICALP), Nafplion, Greece, July 1985.

Manna, Z., and R. Waldinger [85b]

Deduction with relation matching, *5th Conference on Foundations of Software Technology and Theoretical Computer Science*, New Delhi, India (invited paper), Lecture Notes in Computer Science 206, Springer-Verlag, December 1985, pp. 212-224.

Manna, Z., and R. Waldinger [85c]

The origin of the binary-search paradigm, *Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985, pp. 222-224. Also to appear in *Science of Computer Programming.*

Manna, Z., and R. Waldinger [85d]

Plan formation in situational logic, *Workshop on Distributed Artificial Intelligence*, Sea Ranch, CA, December 1985 (invited paper).

Manna, Z., and R. Waldinger [85e]

The Logical Basis for Computer Programming, Addison-Wesley, Reading, MA,

Volume 1: Deductive Reasoning (1985),

Volume 2: Deductive Techniques (to appear).

Jonsson, B., Z. Manna, and R. Waldinger [86]

Towards deductive synthesis of data-flow networks, *First Conference on Logic in Computer Science*, Cambridge, MA (June 1986).

June 1984 Report No. STAN-CS-84-1012

TABLOG: The Deductive-Tableau Programming Language

by

Yonathan Malachi, Zohar Manna and Richard Waldinger

Department of Computer Science

Stanford University
Stanford, CA 94305

The three titles should be processed as
one report.
Per Ms. Debbie Tyrell, AFOSR/XOTD

TABLOG:
The Deductive-Tableau Programming Language

Yonathan Malachi
Zohar Manna

Computer Science Department
Stanford University

Richard Waldinger

Artificial Intelligence Center
SRI International

Abstract

TABLOG (Tableau Logic Programming Language) is a language based on first-order predicate logic with equality that combines functional and logic programming. TABLOG incorporates advantages of LISP and PROLOG.

A program in TABLOG is a list of formulas in a first-order logic (including equality, negation, and equivalence) that is more general and more expressive than PROLOG's Horn clauses. Whereas PROLOG programs must be relational, TABLOG programs may define either relations or functions. While LISP programs yield results of a computation by returning a single output value, TABLOG programs can be relations and can produce several results simultaneously through their arguments.

TABLOG employs the Manna-Waldinger *deductive-tableau* proof system as an interpreter in the same way that PROLOG uses a resolution-based proof system. Unification is used by TABLOG to match a call with a line in the program and to bind arguments. The basic rules of deduction used for computing are nonclausal resolution and rewriting by means of equality and equivalence.

A pilot interpreter for the language has been implemented.

This research was supported in part by the National Science Foundation under Grants MCS-82-14523, MCS-81-11586, and MCS-81-05565, by the United States Air Force Office of Scientific Research under Grant AFOSR-81-0014, by DARPA under Contract N0039-82-C-0250, and by a grant from IBM Research, San Jose, California.

Presented at the ACM Symposium on LISP and Functional Programming, University of Texas at Austin, August 5 8, 1984.

1. Introduction

Logic programming [Kowalski 79] attempts to improve programmer productivity by proposing logic, a human-oriented language, as a programming language. PROLOG, the flagship of logic-programming languages, based on a resolution proof system, has a restricted syntax. TABLOG is based on a more flexible theorem prover, the *deductive-tableau* proof system [Manna and Waldinger 80], which allows a more intuitive and a richer syntax. A TABLOG program is a list of *assertions* in [quantifier-free] first-order logic with equality. The execution of a program corresponds to the proof of a *goal*, which produces the desired output(s) as a side effect.

Since a particular procedure is specified by the programmer, and since the proof taking place is always a proof of a special case of a theorem—namely, the case for the given input—the program interpreter does not need all the deduction rules available in the original deductive-tableau proof system. The theorem prover can be more directed, efficient, and predictable than a theorem prover used for program synthesis or for any other general-purpose deduction.

2. TABLOG Syntax

Syntactic Objects

The language is that of the quantifier-free first-order predicate logic with equality, consisting of the following:

- truth values: *true, false*.
- connectives: $\wedge$, $\vee$, $\neg$, $\equiv$, $\rightarrow$ (*implies*), $\leftarrow$ (*if*), *if-then-else*.
- variables such as u, v, x_1, y_{25}.
- constants such as a, b, $[\,]$, 5.
- predicates such as $=$, **prime**, $\in$, $\geq$.
- functions such as **gcd**, **append**, $+$.

The user must declare the variables, constants, functions, and predicates used in the program; some primitive constants, functions, and predicates (such as 0, $[\,]$, $+$, $-$, $\geq$, **odd**) are predefined.

Note that we use the *if-then-else* construct, both as a connective for formulas

$$\text{if } u = [\,] \text{ then } \mathbf{empty}(u) \text{ else } \mathbf{sorted}(u)$$

and as an operator generating terms

$$\begin{aligned}\mathbf{gcd}(x, y) = \text{if } x \geq y \text{ then } &\mathbf{gcd}(x-y, y)\\ \text{else } &\mathbf{gcd}(x, y-x).\end{aligned}$$

This, together with $\leftarrow$ (reverse implication), enables the programmer to write LISP-style as well as PROLOG-style programs.

Programs

A program is a list of *assertions* (formulas in [quantifier-free] first-order logic with equality), specifying the algorithm. Variables are implicitly universally quantified.

Here is a very simple program for appending two lists:

$$\begin{aligned}\mathbf{append}([\,], v) &= v\\ \mathbf{append}(x \circ u, v) &= x \circ \mathbf{append}(u, v).\end{aligned}$$

The $\circ$ symbol denotes the list insertion (**cons** in LISP) operator, and [] denotes the empty list (**nil** in LISP).

A call to a program is a *goal* to be proved. Like the assertions, goals are formulas in logic, but variables are implicitly existentially quantified. The bindings of these variables are recorded throughout the proof and become the outputs of the program upon termination.

For example, a call to the **append** program above might be

$$z = \mathbf{append}([1,2,3], [a,b]).$$

The output of the execution of this program call will be

$$[1,2,3,a,b],$$

as expected.

The list construct (e.g. [1, 2, 3]) is for convenience in expressing input and output, and denotes the term $1 \circ (2 \circ (3 \circ [\,]))$.

3. Examples

The following examples demonstrate the basic features of TABLOG. The correctness of these programs does not depend on the order of assertions in the program. It is possible, however, to write programs that do take advantage of the known order of the interpreter's goal evaluation, as will be explained later.

In the examples, we use x and y (possibly with subscripts) for variables intended to be assigned atoms (integers in most of the examples); u and v (possibly with subscripts) are variables used for lists.

Deleting a List Element

The following program deletes all [top-level] occurrences of an element x from a list:

$$\begin{aligned}\mathbf{delete}(x, [\,]) &= [\,]\\ \mathbf{delete}(x, y \circ u) &= (\text{if } x = y \text{ then } \mathbf{delete}(x, u)\\ &\qquad \text{else } y \circ \mathbf{delete}(x, u)).\end{aligned}$$

This program demonstrates the use of equality, *if-then-else*, and recursive calls. For those who prefer the PROLOG style of programming, the last line could be replaced by assertions:

$$\begin{aligned}&\mathbf{delete}(x,\ x \circ u) = \mathbf{delete}(x, u)\\ &x \neq y \ \rightarrow\ \mathbf{delete}(x,\ y \circ u) = y \circ \mathbf{delete}(x, u)\end{aligned}$$

To remove all occurrences of a from the list $[a, b, a, c]$ the goal

$$z = \mathbf{delete}(a,\ [a, b, a, c])$$

is given to the interpreter.

Set Union

The following example, a program to find the union of two sets represented by lists, demonstrates the use of negation, equivalence and *if-then-else*:

1. $\mathbf{union}([\,], v) = v$
2. $\mathbf{union}(x \circ u, v) \ = \text{if } \mathbf{member}(x, v)$
   $\text{then } \mathbf{union}(u, v)$
   $\text{else } (x \circ \mathbf{union}(u, v))$
3. $\neg\mathbf{member}(x, [\,])$
4. $\mathbf{member}(x,\ y \circ u) \ \equiv\ ((x = y) \vee \mathbf{member}(x, u))$

Lines 1 and 2 define the **union** function. Line 1 defines the union of the empty set with another set, and line 2 asserts that the head x of the first set $x \circ u$ should be inserted into the union if it is not already in the second set v.

Lines 3 and 4 define the **member** relation. Line 3 specifies that no element is a member of the empty set, and line 4 defines how to test recursively membership in a nonempty set.

Factorial

The following program will compute the factorial of a nonnegative integer x:

$$\begin{aligned}&\mathbf{fact}(0) = 1\\ &\mathbf{fact}(x) = x * \mathbf{fact}(x - 1) \ \leftarrow\ x \geq 1\end{aligned}$$

The corresponding PROLOG program will be

$$\begin{aligned}&\mathbf{factp}(0, 1)\\ &\mathbf{factp}(x, z) \ \leftarrow\ x_1 \ \mathbf{is}\ x - 1 \ \wedge\ \mathbf{factp}(x_1, y) \ \wedge\ z \ \mathbf{is}\ x * y.\end{aligned}$$

The **is** construct is used in PROLOG to force the evaluation of an arithmetic expression.

Quicksort

Here is a TABLOG program that uses quicksort to sort a list of numbers. It combines a PROLOG-style relational subprogram for partitioning with a LISP-style functional subprogram for sorting.

1. $\mathbf{qsort}([\,]) = [\,]$

2. $\mathbf{qsort}(x \circ u) = \mathbf{append}(\mathbf{qsort}(u_1),\ x \circ \mathbf{qsort}(u_2))$
   $\quad \leftarrow\ \mathbf{partition}(x, u, u_1, u_2)$

3. $\mathbf{partition}(x, [\,], [\,], [\,])$

4. $\mathbf{partition}(x, y \circ u, y \circ u_1, u_2)$
   $\quad \leftarrow\ y \leq x\ \wedge\ \mathbf{partition}(x, u, u_1, u_2)$

5. $\mathbf{partition}(x, y \circ u, u_1, y \circ u_2)$
   $\quad \leftarrow\ y > x\ \wedge\ \mathbf{partition}(x, u, u_1, u_2)$

The assertions in lines 1 and 2 form the sorting subprogram. Line 1 asserts that the empty list is already sorted. Line 2 specifies that, to sort a list $x \circ u$, with head x and tail u, one should append the sorted version of two sublists of u, u_1 and u_2, and insert the element x between them; the two sublists u_1 and u_2 are determined by the subprogram **partition** to be the elements of u less than or equal to x and greater than x, respectively.

The assertions in lines 3 to 5 specify how to partition a list according to a partition element x. Line 3 discusses the partitioning of the empty list, while lines 4 and 5 treat the case in which the list is of the form $y \circ u$. Line 4 is for the case in which y, the head of the list, is less than or equal to x; therefore, y should be inserted into the list u_1 of elements not greater than x. Line 5 is for the alternative case.

The **append** function for appending two lists was defined earlier.

4. Comparison with PROLOG

Functions and Equality

While PROLOG programs must be relations, TABLOG programs can be either relations or functions. The availability of functions and equality makes it possible to write programs more naturally. The functional style of programs frees the programmer from the need to introduce many auxiliary variables.

We can compare the PROLOG and TABLOG programs for quicksort. In TABLOG, the program uses the unary function **qsort** to produce a value, whereas a PROLOG program is a binary relation **qsortp**: the second argument is needed to hold the output.

The second assertion in the TABLOG program is

$$\mathbf{qsort}(x \circ u) = \mathbf{append}(\mathbf{qsort}(u_1), x \circ \mathbf{qsort}(u_2))$$
$$\quad \leftarrow\ \mathbf{partition}(x, u, u_1, u_2)$$

The corresponding clause in the PROLOG program will be something like

$$\begin{aligned}\mathbf{qsortp}(x \circ u, z) \ \leftarrow\ & \mathbf{partition}(x, u, u_1, u_2) \wedge \\ & \mathbf{qsortp}(u_1, z_1) \wedge \\ & \mathbf{qsortp}(u_2, z_2) \wedge \\ & \mathbf{appendp}(z_1, x \circ z_2, z).\end{aligned}$$

The additional variables z_1 and z_2 are required to store the results of sorting u_1 and u_2. This demonstrates the advantage of having functions and equality in the language. Note that, although function symbols exist in PROLOG, they are used only for constructing data structures (like TABLOG's primitive functions) and are not reduced.

Negation and Equivalence

In PROLOG, negation is not available directly; it is simulated by finite failure. To prove *not*(P), PROLOG attempts to prove P; *not*(P) succeeds if and only if the proof of P fails. In TABLOG, negation is treated like any other connective of logic. Therefore, we can prove formulas such as $\neg\mathbf{member}(1, [2, 3])$.

The TABLOG **union** program, described earlier, uses both equivalence and negation:

$$\mathbf{union}([\,], v) = v$$

$$\begin{aligned}\mathbf{union}(x \circ u, v) = \ & \text{if } \mathbf{member}(x, v) \\ & \text{then } \mathbf{union}(u, v) \\ & \text{else } (x \circ \mathbf{union}(u, v))\end{aligned}$$

$$\neg\mathbf{member}(x, [\,])$$

$$\mathbf{member}(x, y \circ u) \equiv (x = y) \vee \mathbf{member}(x, u).$$

Here is a possible PROLOG implementation of the same algorithm:

$$\mathbf{unionp}(x \circ u, v, z) \ \leftarrow \mathbf{memberp}(x, v) \wedge \mathbf{unionp}(u, v, z)$$

$$\mathbf{unionp}(x \circ u, v, x \circ z) \leftarrow \mathbf{unionp}(u, v, z)$$

$$\mathbf{unionp}([\,], v, v)$$

$$\mathbf{memberp}(x, x \circ u)$$

$$\mathbf{memberp}(x, y \circ u) \leftarrow \mathbf{memberp}(x, u).$$

Changing the order of the first two clauses in the PROLOG program will result in an incorrect output; the second clause is correct only for the case in which x is not a member of v. The TABLOG assertions can be freely rearranged; this suggests that all of them can be matched against the current goal in parallel, if desired.

Unification

The unification procedure built into PROLOG is not really unification (e.g., as defined in [Robinson 65]); it does not fail in matching an expression against one of its proper subexpressions since it lacks an *occur-check*. When a theorem prover is used as a program interpreter, the omission of the occur-check makes it possible to generate cyclic expressions that may not correspond to any concrete objects.

The unification used by the TABLOG interpreter does include an occur-check, so that only theorems can indeed be proved.

5. Comparison with LISP

LISP programs are functions, each returning one value; the arguments of a function must be bound before the function is called. In TABLOG, on the other hand, programs can be either relations or functions, and the arguments need not be bound; these arguments will later be bound by unification.

We can illustrate this with the quicksort program again, concentrating on the partition subprogram. In TABLOG, we have seen how to achieve the partition by a predicate with four arguments, two for input and two for output:

```
1. **partition**$(x, [\,], [\,], [\,])$
2. **partition**$(x, y \circ u, y \circ u_1, u_2)$
       $\leftarrow\ y \le x\ \wedge$ **partition**$(x, u, u_1, u_2)$
3. **partition**$(x, y \circ u, u_1, y \circ u_2)$
       $\leftarrow\ y > x\ \wedge$ **partition**$(x, u, u_1, u_2)$
```

The definition of the program **partition** is much shorter and cleaner than the corresponding LISP program:

```
highpart(x, u)  ⇐
    if null(u) then nil
    else-if $x \ge$ car(u) then highpart(x, cdr(u))
    else cons(car(u), highpart(x, cdr(u)))

lowpart(x, u)  ⇐
    if null(u) then nil
    else-if $x \ge$ car(u)
        then cons(car(u), lowpart(x, cdr(u)))
    else lowpart(x, cdr(u)).
```

We can generate the two sublists in LISP simultaneously, but this will require even more pairing and decomposition.

Note that unification also gives us "free" decomposition of the list argument into its head and tail; in the LISP program, this decomposition requires explicit calls to the functions **car** and **cdr**.

6. The Deductive-Tableau Proof System

In this section, we give a brief summary of the Manna-Waldinger deductive-tableau proof system [Manna and Waldinger 80 and 82]. This proof system is used as the TABLOG interpreter. We describe only the deduction rules actually employed in it.

A *deductive tableau* consists of rows, each containing either an *assertion* or a *goal*. The assertions and goals (both of which we refer to by the generic name *entries*) are first-order logic formulas; the theorem is proved by manipulating them. The declarative or logical meaning of a tableau is that, if every instance of all the assertions is true, then some instance of at least one of the goals is true. The assertions in the tableau are like clauses in a standard resolution theorem prover—but they can be arbitrary first-order formulas, not just disjunctions of literals.

The theorem to be proved is entered as the initial goal. A proof is constructed by adding new goals to the tableau, using deduction rules, in such a way that the final tableau is semantically equivalent to the original one. The proof is complete when we have generated the goal *true*.

Deduction Rules

The basic rules used for the program execution task are the following:

- *Nonclausal Resolution*: This generalized resolution rule allows removal of a subformula P from a goal $\mathcal{G}[P]$ by means of an appropriate assertion $\mathcal{A}[\hat{P}]$. Resolving the goal

  $$\mathcal{G}[P]$$

  with the assertion

  $$\mathcal{A}[\hat{P}],$$

  provided that P and $\hat{P}$ are unifiable, i.e., $P\theta = \hat{P}\theta$ for some (most-general) unifier θ, we get the new goal

  $$not(\mathcal{A}'[false]) \ \wedge \ \mathcal{G}'[true],$$

  where $\mathcal{A}'[false]$ is $\mathcal{A}\theta$ after all occurrences of $P\theta$ have been replaced by *false*, and similarly for $\mathcal{G}'[true]$. This deduction rule can be justified by case analysis.

  The choice of the unified subformulas is governed by the *polarity strategy* [Murray 82]. A subformula has *positive* polarity if it occurs within an even number of (explicit or implicit) negations, and has *negative* polarity if it occurs within an odd number of negations. (An assertion has an implicit negation applied to it.) A subformula can occur both positively and negatively in a formula. According to the polarity strategy, the subformula P will be replaced by *false* only if it occurs with negative polarity and the subformula Q will be replaced by *true* only if it occurs with positive polarity.

- *Equality Rule*: An asserted [possibly conditional] equality of two terms can be used to replace one of the terms with the other in a goal. If the asserted equality is conditional, the conditions are added to the resulting goal as conjuncts.

Thus, suppose the assertion is of the form

$$\mathcal{A}[s = t],$$

and the goal is

$$\mathcal{G}[\hat{s}],$$

where s and $\hat{s}$ are unifiable, i.e., $s\theta = \hat{s}\theta$ for some unifier θ. Then we get the new goal

$$not(\mathcal{A}'[false]) \ \wedge \ \mathcal{G}'[t'],$$

where $\mathcal{A}'[false]$ is $\mathcal{A}\theta$ after all occurrences of the equality $s\theta = t\theta$ (which should occur with negative polarity) have been replaced by *false*, and where $\mathcal{G}[t']$ is $\mathcal{G}\theta$ after the replacement of all occurrences of the term $s\theta$ by $t\theta$.

The reflexivity axiom for equality $x = x$ is implicitly included among the assertions of every tableau.

- *Equivalence Rule*: The replacement of one subformula by another asserted to be equivalent to it. This is completely analogous to the equality rule except that we replace atomic formulas rather than terms, using equivalence rather then equality.
- *Simplification*: The replacement of a formula by an equivalent but simpler formula. Both propositional and arithmetic simplification are performed automatically by the TABLOG interpreter.

While nonclausal resolution and the equivalence rule can be performed unifying arbitrary subformulas, the TABLOG interpreter applies these deduction rules unifying atomic subformulas only.

7. Program Semantics

The logical interpretation of a tableau containing a TABLOG program and a call to it is the logical sentence associated with the tableau: the conjunction of the universal closures of the assertions implies the existential closure of the goal.

The desired goal is reduced to *true* by means of the assertions and the deduction rules. The variables are bound when subexpressions of the goal (or derived subgoals) are unified with subexpressions of the assertions. The order of the reduction is explained in the next section. The output of the program is the final binding of the variables of the original goal.

We distinguish between *defined* functions, whose semantics is defined by the user program, and *primitive* functions, which are either data constructors (e.g., $\circ$), or are built-in and have their semantics defined by attached procedures in the simplifier; for example, an expression like $(2 + x + 5) \circ [\,]$ is considered primitive and will be automatically simplified to $(x + 7) \circ [\,]$.

As in PROLOG, variables are local to the assertion or goal in which they appear. Renaming of variables is done automatically by the interpreter when there is a collision of names between the goal and assertion involved in a derivation step.

The variables of the original goal are the output variables. The interpreter keeps their binding throughout the derivation; the same variable name can be used for a different purpose in other assertions or goals.

8. Program Execution

Every line in a program is an assertion in the tableau; a call to the program is a goal in the same tableau.

The tableau system provides us with deduction rules but with no specific order in which to apply them. To use it as a programing language, we have to specify the order of application both for predictability and for efficiency.

The proof system is used to execute programs in a way analogous to the inversion of a matrix by linear operations on its rows, where we simultaneously apply the same transformations to the matrix to be inverted and to the identity matrix. In the program execution process, we start with a tableau containing the assertions of the program and a goal calling this program; we apply the same substitutions (obtained by unification) to the current subgoal and to the binding of the output variables. A matrix inversion is complete when we reduce the original matrix to the identity matrix: in TABLOG we are done when we have reduced the original goal to *true*. At this point, the result of the computation is the final binding of the output variables.

Although in the declarative (logical) semantics of the tableau the order of entries is immaterial, the procedural interpretation of the tableau as a program takes this order into account; changing the order of two assertions or changing the order of the conjuncts or disjuncts in an assertion or a goal may produce different computations.

The user for his part, has to specify an algorithm by employing the predefined order of evaluation of the tableau. At each step of the execution, one *basic expression* (a nonvariable term or an atomic formula) of the current goal is reduced. The expression to be reduced is selected by scanning the goal from left to right. The first (leftmost) basic expression that has only primitive arguments (i.e., that contain only variables, constants, and primitive functions) is chosen and reduced, if possible. Matching the selected expression against assertions is done in order of appearance.

This is best explained with an example:

To sort the list $[2, 1, 4, 3]$ using quicksort, we write the goal

$$z = \mathbf{qsort}([2,1,4,3]).$$

To execute this goal, the expression chosen for reduction will be the term $\mathbf{qsort}([2,1,4,3])$, i.e., $\mathbf{qsort}(2 \circ [1,4,3])$. This term unifies with the leftmost term $\mathbf{qsort}(x \circ u)$ in the second assertion of the quicksort program,

$$\begin{aligned} \mathbf{qsort}(x \circ u) = {} & \mathbf{append}(\mathbf{qsort}(u_1), x \circ \mathbf{qsort}(u_2)) \\ & \leftarrow \ \mathbf{partition}(x, u, u_1, u_2). \end{aligned}$$

According to the equality rule, it will be replaced by the corresponding instance of the right-hand side of the equality; this is done only after the unifier

$$\{x \leftarrow 2, \ u \leftarrow [1,4,3]\}$$

is applied to both the goal and the assertion. The occurrence of the equality

$$\mathbf{qsort}(2 \circ [1,4,3]) = \mathbf{append}(\mathbf{qsort}(u_1), 2 \circ \mathbf{qsort}(u_2))$$

is replaced by *false* in the [modified] assertion, the occurrence of the term

$$\mathbf{qsort}(2 \circ [1,4,3])$$

is replaced by the term

$$\mathbf{append}(\mathbf{qsort}(u_1), 2 \circ \mathbf{qsort}(u_2))$$

in the (modified) goal, and a conjunction is formed, obtaining

$$\begin{aligned}&not(false \leftarrow \ \mathbf{partition}(2, [1,4,3], u_1, u_2) \ \wedge \\ &z = \mathbf{append}(\mathbf{qsort}(u_1), 2 \circ \mathbf{qsort}(u_2)).\end{aligned}$$

This formula can be reduced by the simplifications

$$(false \leftarrow P) \ \Rightarrow \ not \ P$$

and

$$not(not \ P) \ \Rightarrow \ P$$

to obtain the new goal

$$\begin{aligned}&\mathbf{partition}(2, [1,4,3], u_1, u_2) \ \wedge \\ &z = \mathbf{append}(\mathbf{qsort}(u_1), 2 \circ \mathbf{qsort}(u_2)).\end{aligned}$$

Continuing with this example, we now have a case in which the expression to be reduced is an atomic formula, namely,

$$\mathbf{partition}(2, [1,4,3], u_1, u_2).$$

This atomic formula is unifiable with a subformula in the second assertion of the **partition** subprogram (with variables renamed to resolve collisions)

$$\begin{aligned}&\mathbf{partition}(x, y \circ u, y \circ u_3, u_4) \\ &\qquad \leftarrow \ y \le x \ \wedge \ \mathbf{partition}(x, u, u_3, u_4).\end{aligned}$$

Nonclausal resolution is now performed to further reduce the current goal. The unifier

$$\{x \leftarrow 2, \ y \leftarrow 1, \ u \leftarrow [4,3], \ u_1 \leftarrow 1 \circ u_3, \ u_2 \leftarrow u_4\}$$

is applied to both the assertion and the goal; the formula

$$\mathbf{partition}(2, [1,4,3], 1 \circ u_3, u_4)$$

is replaced by *false* in the [modified] assertion and by *true* in the goal. Once again a conjunction is formed and the new goal generated (after simplification) is

$$\begin{aligned}&\mathbf{partition}(2, [4, 3], u_3, u_4) \wedge\\ &z = \mathbf{append}(\mathbf{qsort}(1 \circ u_3), 2 \circ \mathbf{qsort}(u_4)).\end{aligned}$$

Eventually we reach the subgoal

$$z = [1, 2, 3, 4],$$

where the right-hand side of the equality contains only primitive functions and constants. The execution then terminates and the desired output is

$$[1, 2, 3, 4].$$

Note that some functions and predicates (e.g., $\circ$ in this example) are predefined to be primitive: an expression in which such a symbol is the main operator is never selected to be reduced, although its subexpressions may be reduced.

Backtracking

If the selected expression cannot be reduced, the search for other possible reductions is done by backtracking.

In PROLOG each goal is a conjunction, so all the conjuncts must be proved; this means that, when facing a dead end, we have to undo the most recent binding and try other assertions.

In TABLOG the situation is more complex: each goal (and each assertion) is an arbitrary formula, so it is possible to satisfy it without satisfying all its atomic subformulas. Therefore, when the TABLOG *interpreter fails to find an assertion that reduces some basic* expression, it tries to reduce the next expression that can allow the proof to proceed. In the case in which the expression that cannot be reduced is "essential" (for example, a conjunct in a conjunctive goal), no other subexpression will be attempted and backtracking will occur.

During backtracking, the goal from which the current goal was derived becomes the new current goal, but the next plausible assertion is used. This is similar to the backtracking used in PROLOG.

The Implementation

A prototype interpreter for TABLOG is implemented in MACLISP. The implemented system serves as a program editor, debugger, and interpreter. All the examples mentioned in this paper have been executed on this interpreter.

The backtracking mechanism provides a simple way of changing the interpreter so that lazy evaluation can be employed - i.e., so that attempts can be made to evaluate expressions even if they have nonprimitive arguments.

Because the interpreter is built on top of a versatile theorem-proving system, the execution of programs is relatively slow. The interpreter now handles complicated cases that might arise in a more general theorem-proving task, but will never occur in TABLOG. We hope that performance will be improved considerably by tuning the simplifier and utilizing tricks from PROLOG implementations to make the binding of variables faster.

9. Related Research

Logic programming has become a fashionable research topic in recent years. Most of the research relates to PROLOG and its extensions. We mention here some of the work that has been done independently of TABLOG to generate languages similar to TABLOG in their intention and capabilities.

While the deductive-tableau theorem prover used for TABLOG execution is based on a generalized resolution inference rule, [Haridi 81], [Haridi and Sahlin 83], and [Hansson, Haridi, and Tärnlund 82] describe a programming language based on a natural-deduction proof system. They do allow quantifiers and other connectives in the language but the syntax of their assertions is somewhat restricted.

[Kornfeld 83] extends PROLOG to include equality; asserting equality between two objects in his language causes the system to unify these objects when regular unification fails. This makes it possible to unify objects that differ syntactically. Kornfeld treats only Horn clauses and does not introduce any substitution rule either for equality or for equivalence.

[Tamaki 84] extends PROLOG by introducing a reducibility predicate, denoted by $\triangleright$. This predicate has semantics similar to the way TABLOG uses equality for rewriting terms. This work also includes *f-symbols* and *d-symbols* that are analogous to TABLOG's distinction between defined and primitive functions. The possible nesting of terms is restricted and programs must be in Horn clause form.

OBJ [Goguen, Meseguer, and Plaisted 82] is also related to logic programming. It is based, however, on the algebraic semantics of abstract data types and equational theory rather than on [resolution-based] theorem proving in first-order logic. OBJ1 is an advanced implementation of the language that allows parameterized and hierarchical programming. OBJ1 includes system features for convenience and efficiency; it uses one-way pattern matching to apply rewrite rules rather than two-way unification. [Goguen and Meseguer 84] describes EQLOG, the extension of OBJ to include unification and Horn clauses.

There are PROLOG systems, such as LOGLISP [Robinson and Sibert 82] and QLOG [Komorowski 79 and 82] that are implemented within LISP systems. These systems allow the user to invoke the PROLOG interpreter from within a LISP program and vice versa. In TABLOG, however, LISP-like features and PROLOG-like features coexist peacefully in the same framework and are processed by the same deductive engine.

10. Conclusions and Discussion

The TABLOG language is a new approach to logic programming: instead of patching up PROLOG with new constructs to eliminate its shortcomings, we suggest a more powerful deductive engine.

The combination in TABLOG of unification as a binding mechanism, equality for specifying functions, and first-order logic for specifying predicates creates a rich language that is clean from a logical point of view. As a consequence, programs correspond to our intuition and are easier to write, read, and modify. We can mix LISP-style and PROLOG-style programming and use whichever is more convenient for the problem or subproblem.

By restricting the general-purpose deductive-tableau theorem prover and forcing it to follow a specific search order, we have made it suitable to serve as a program interpreter; the specific search order makes it both more predictable and more efficient than attempting to apply the deduction rules arbitrarily.

While the theorem prover supports reasoning with quantified formulas [Manna and Waldinger 82; Bronstein 83], the ramifications of including quantifiers in the language are still under investigation. Quantifiers would certainly enhance the expressive power of TABLOG, but we believe that they are more suited to a specification language than a programming language.

It seems very natural to extend TABLOG to parallel computation. The inclusion of real negation makes it possible to write programs that do not depend on the order of assertions.

The extension of TABLOG to support concurrent programs is being pursued. If the conditions of the assertions are disjoint, several assertions can be matched against the current subgoal in parallel. In addition, disjunctive goals can be split between processes. If there are no common variables, conjuncts can be solved in parallel; otherwise some form of communication is required.

The *or-parallelism* and *and-parallelism* suggested for PROLOG are applicable for TABLOG as well. The or-parallelism of PROLOG relates to matching against many assertions; in TABLOG or-parallelism is possible within every goal, since, for example, goals can be disjunctive. In TABLOG can other forms of parallelism can be applied to nested function calls.

Acknowledgments

Thanks are due to Martin Abadi, Yoram Moses, Oren Patashnik, Jon Traugott, and Joe Weening for comments on various versions of this paper. We are especially indebted to Bengt Jonsson and Frank Yellin for reading many versions of the manuscript and providing insightful comments and suggestions.

References

[Bronstein 83]
A. Bronstein, "Full quantification and special relations in a first-order logic theorem prover," programming project, Computer Science Department, Stanford University, 1983.

[Clark and Tärnlund 82]
K. L. Clark and S.-Å. Tärnlund (editors), *Logic Programming*, Academic Press (1982). A.P.I.C. Studies in Data Processing No. 16.

[Goguen and Meseguer 84]
J. Goguen and J. Meseguer, "Equality, types, modules and generics for logic programming," in *Proceedings of the Second International Logic Programming Conference*, Uppsala, Sweden, July 2-6, 1984.

[Goguen, Meseguer, and Plaisted 82]
J. Goguen, J. Meseguer, and D. Plaisted, "Programming with parameterized abstract objects in OBJ," in *Theory and Practice of Software Technology*, edited by D. Ferrari, M. Bolognani, and J. Goguen, North-Holland, 1982.

[Hansson, Haridi, and Tärnlund 82]
A. Hansson, S. Haridi, and S.-Å. Tärnlund, "Properties of a Logic Programming Language," in [Clark and Tärnlund 82].

[Haridi 81]
S. Haridi, "Logic programming based on a natural deduction system," Ph.D. Thesis, Department of Telecommunication Systems and Computer Science, The Royal Institute of Technology, Stockholm, Sweden, 1981.

[Haridi and Sahlin 83]
S. Haridi and D. Sahlin, "Evaluation of logic programs based on natural deduction," Technical report RITA-CS-8305 B, Department of Telecommunication Systems and Computer Science, The Royal Institute of Technology, Stockholm, Sweden, 1983.

[Komorowski 79]
H. J. Komorowski, "The QLOG Interactive Environment," Technical Report LITH-MAR-R-79-19, Informatics Lab, Linkopping University, Sweden, August 1979.

[Komorowski 82]
H. J. Komorowski, "QLOG The Programming Environment for Prolog in LISP," in [Clark and Tärnlund 82]

[Kornfeld 83]
W. Kornfeld, "Equality for Prolog," in *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, August 1983.

[Kowalski 79]
R. Kowalski, *Logic for Problem Solving*, North-Holland, 1979.

[Manna and Waldinger 80]
Z. Manna and R. Waldinger, "A deductive approach to program synthesis," *ACM Transactions on Programming Languages and Systems*, Vol. 2, No. 1, pp. 92 121, January 1980.

[Manna and Waldinger 82]
Z. Manna and R. Waldinger, "Special relations in program-synthetic deduction," Department of Computer Science, Technical Report No. STAN-CS-82-902, Stanford University. To appear in *Journal of the ACM*.

[Murray 82]
N. V. Murray, "Completely nonclausal theorem proving," *Artificial Intelligence*, Vol. 18, No. 1, pp. 67 85.

[Robinson 65]
J. A. Robinson, "A machine-oriented logic based on the resolution principle," *Journal of the ACM*, Vol. 12, No. 1, Jan 1965, pp. 23 41.

[Robinson and Sibert 82]
J. A. Robinson and E. E. Sibert, "LOGLISP: and alternative to PROLOG," in *Machine Intelligence 10*, J. E. Hayes, D. Michie, and Y-H Pao editors, Ellis Horwood Ltd., Chichester, 1982.

[Tamaki 84]
H. Tamaki, "Semantics of a logic programming language with a reducibility predicate," *Proceedings of the IEEE Logic Programming Conference*, Atlantic City, February 1984.

March 1985 Report No. STAN-CS-85-1044

The Origin of the Binary-Search Paradigm

by

Zohar Manna
Richard Waldinger

Department of Computer Science

Stanford University
Stanford, CA 94305

THE ORIGIN OF THE BINARY-SEARCH PARADIGM

ZOHAR MANNA
Computer Science Department
Stanford University
Stanford, CA 94305

RICHARD WALDINGER
Artificial Intelligence Center
SRI International
Menlo Park, CA 94025

ABSTRACT

In a binary-search algorithm for the computation of a numerical function, the interval in which the desired output is sought is divided in half at each *iteration*. The paper considers how such algorithms might be derived from their specifications by an automatic program-synthesis system. The derivation of the binary-search concept has been found to be surprisingly straightforward. The programs obtained, though reasonably simple and efficient, are quite different from those that would have been constructed by informal means.

Key Words: program synthesis, theorem proving, binary search, real square root

INTRODUCTION

Some of the most efficient algorithms for the computation of numerical functions rely on the technique of *binary search*; according to this technique, the interval in which the desired output is sought is divided in half at each iteration until it is smaller than a given tolerance.

For example, let us consider the following program for finding a real-number approximation to the square root of a nonnegative real number r. The program sets z to be within a given positive tolerance ϵ less than $\sqrt{r}$.

```
$z \leftarrow 0$
$v \leftarrow max(r,\ 1)$
while $\epsilon \leq v$ do $v \leftarrow v/2$
                    $if\ [z+v]^2 \leq r\ then\ z \leftarrow z+v$
$return(z)$
```

This is a classical square-root program based on one that appeared in Wensley [59]. The program establishes and maintains the loop invariant that z is within v less than $\sqrt{r}$, i.e., that $\sqrt{r}$ belongs to the half-open interval $[z,\ z+v)$. At each iteration, the program divides this interval in half and tests whether $\sqrt{r}$ is in the right or left half, adjusting z and v accordingly, until v is smaller than

This research was supported in part by the National Science Foundation under grants MCS-82-14523 and MCS-81-05565, by Defense Advanced Research Projects Agency under Contract N00039-84-C-0211, by the United States Air Force Office of Scientific Research under Contract AFOSR-81-0014, by the Office of Naval Research under Contract N00014-84-C-0706, and by a contract from the International Business Machines Corporation.

the given tolerance ϵ. The program is reasonably efficient; it terminates after $\lceil log_2(max(r, 1)/\epsilon)\rceil$ iterations.

Analogous programs provide an efficient means of computing a variety of numerical functions. It is not immediately obvious how such programs can be developed by *automatic program-synthesis* systems, which derive programs to meet given specifications. Some researchers (e.g., Dershowitz and Manna [77], Smith [85]) have suggested that synthesis systems be provided with several general program schemata, which could be specialized as required to fit particular applications. Binary search would be one of these schemata. The system would be required to discover which schema, if any, is applicable to a new problem.

It may indeed be valuable to provide a synthesis system with general schemata, but this approach leaves open the question of how such schemata are discovered in the first place. To our surprise, we have found that the concept of binary search emerges quite naturally and easily in the derivations of some numerical programs and does not need to be built in. The programs we have obtained in this way are reasonably simple and efficient, but bizarre in appearance and quite different from those we would have constructed by informal means.

The programs have been derived in a deductive framework (Manna and Waldinger [80], [85]) in which the process of constructing a program is regarded as a task of proving a mathematical theorem. According to this approach, the program's specification is phrased as a theorem, the theorem is proved, and a program guaranteed to meet the specification is extracted from the proof. If the specification reflects our intentions correctly, no further verification or testing is required.

In this paper we outline our deductive framework and show the derivation of a numerical program up to the point at which the binary-search concept emerges. We then show several analogous binary-search programs that have been developed by this method. Finally we discuss what these findings indicate about the prospects for automatic program synthesis.

DEDUCTIVE PROGRAM SYNTHESIS

In this section we describe our framework for deductive program synthesis, emphasizing those aspects that are essential for the derivation fragment that appears in this paper. Readers who would like a fuller introduction to this approach are referred to Manna and Waldinger ([80], [85]).

We begin with an outline of the logical concepts we shall need.

LOGICAL PREREQUISITES

The system deals with

- *terms* composed (in the usual way) of *constants* $a, b, c, \ldots$, variables $u, v, w, \ldots$, function symbols, and the conditional (*if-then-else*) term constructor.

- *atoms* composed of terms, relation (predicate) symbols, including the equality symbol =, and the truth symbols *true* and *false*.

- *sentences* composed of atoms and logical connectives.

Sentences are quantifier-free. We sometimes use infix notation for function and relation symbols (for example, $x + a$ or $0 \leq y$). An *expression* is a term or a sentence. An expression is said to be *ground* if it contains no variables. Certain of the symbols are declared to be *primitive*: these are the computable symbols of our programming language.

Let e, s, and t be expressions, where s and t are either both sentences or both terms. If we write e as $e[s]$, then $e[t]$ denotes the result of replacing every occurrence of s in $e[s]$ with t.

We loosely follow the terminology of Robinson [79]. We denote a substitution θ by $\{x_1 \leftarrow t_1, x_2 \leftarrow t_2, \ldots, x_n \leftarrow t_n\}$. For any expression e, the expression $e\theta$ is the result of *applying θ to e*, obtained by simultaneously replacing every occurrence of the variable x_i in e with the corresponding term t_i. We shall also say that $e\theta$ is an *instance* of e.

Variables in sentences are given an implicit universal quantification; a sentence is true under a given interpretation if and only if every instance of the sentence is true, and if and only if every ground instance of the sentence (i.e., an instance that contains no variables) is true.

Let e, s, and t be expressions, where s and t are either both sentences or both terms, and let θ be a substitution. If we write e as $e[s]$, then $e\theta[t]$ denotes the result of replacing every occurrence of $s\theta$ in $e\theta$ with t.

We now describe the basic notions of deductive program synthesis.

SPECIFICATIONS AND PROGRAMS

A specification is a statement of the purpose of the desired program, which need give no indication of the method by which that purpose is to be achieved. In this paper we consider only *applicative* (or *functional*) programs, which yield an output but alter no data structures and produce no other side effects. The specifications for these programs have the form

$$\begin{aligned} f(a) \;\Leftarrow\; & \text{find } z \text{ such that } \mathcal{R}[a,\, z] \\ & \text{where } \mathcal{P}[a]. \end{aligned}$$

In other words, the program f we want to construct is to yield, for a given *input* a, an *output* z satisfying the *output condition* $\mathcal{R}[a, z]$, provided that the input a satisfies the *input condition* $\mathcal{P}[a]$. In other words, z is to satisfy the *input-output* condition

$$\begin{aligned} & \textit{if } \mathcal{P}[a] \\ & \textit{then } \mathcal{R}[a,\, z]. \end{aligned}$$

For example, suppose we want to specify the program *sqrt* to yield a real number z that is within a given tolerance ϵ less than $\sqrt{r}$, the exact square root of a given nonnegative real number r. Then we might write

$$\begin{aligned} sqrt(r,\, \epsilon) \;\Leftarrow\; & \text{find } z \text{ such that} \\ & \qquad z^2 \leq r \;\; \textit{and} \;\; \textit{not}\; \left[(z+\epsilon)^2 \leq r\right] \\ & \text{where } 0 \leq r \;\; \textit{and} \;\; 0 < \epsilon. \end{aligned}$$

In other words, we want to find an output z satisfying the output condition

$$z^2 \leq r \;\; \mathit{and} \;\; \mathit{not} \; \left[(z + \epsilon)^2 \leq r\right],$$

provided that the inputs r and ϵ satisfy the input condition

$$0 \leq r \;\; \mathit{and} \;\; 0 < \epsilon.$$

The above square-root specification is not a program and does not indicate a particular method for computing the square root; it describes the input-output behavior of many programs, employing different algorithms and perhaps producing different outputs.

The programs we consider are sets of expressions of the form

$$f_i(a) \Leftarrow t_i,$$

where t_i is a *primitive* term, i.e., one expressed entirely in the vocabulary of our programming language. These programs can be mutually recursive; i.e., we regard the function symbols f_i as primitive. In the usual way, such a program indicates a method for computing an output. For the most part, in this paper we shall consider programs consisting of only a single expression $f(a) \Leftarrow t$, which may be recursive.

In a given theory, a program f is said to *satisfy* a specification of the above form if, for any input a satisfying the input condition $\mathcal{P}[a]$, the program $f(a)$ terminates and produces an output t satisfying the output condition $\mathcal{R}[a,\ t]$.

DEDUCTIVE TABLEAUS

The fundamental structure of our system, the deductive tableau, is a set of *rows*, each of which must contain a sentence, either an *assertion* or a *goal*; any of these rows may contain an expression, the *output entry*. An example of a tableau follows:

assertions	goals	outputs $f(a)$
$\mathcal{P}[a]$		
	$\mathcal{R}[a,\ z]$	z
if $q(u)$ *then* $\mathcal{R}[u,\ 0]$		
	$q(a)$	0

Here u and z are variables and a and 0 are constants.

Under a given interpretation, a tableau is true whenever the following condition holds:

> If all instances of each of the assertions are true,
> then some instance of at least one of the goals is true.

Equivalently, the tableau is true if some instance of at least one of the assertions is false or some instance of at least one of the goals is true. Thus, the above tableau is true if $\mathcal{P}[a]$ is false, if

$$\begin{array}{l} \textit{if } q(b) \\ \textit{then } \mathcal{R}[b,\ 0] \end{array}$$

is false, if $\mathcal{R}[a,\ c]$ is true, or if $q(a)$ is true (among other possibilities).

In a given theory, a tableau is said to be *valid* if it is true under any model for the theory.

Under a given interpretation and for a given specification

$$\begin{array}{l} f(a) \ \Leftarrow\ \text{find } z \text{ such that } \mathcal{R}[a, z] \\ \quad \text{where } \mathcal{P}[a], \end{array}$$

a goal is said to have a *suitable* output entry if, whenever an instance of the goal is true, the corresponding instance t' of the output entry will satisfy the *input-output* condition

$$\begin{array}{l} \textit{if } \mathcal{P}[a] \\ \textit{then } \mathcal{R}[a,\ t']. \end{array}$$

(If the goal has no explicit output entry, then it is said to have a suitable output entry if, whenever an instance of the goal is true, any term t' satisfies the input-output condition.) An assertion is said to have a suitable output entry if, whenever an instance of the assertion is false, the corresponding instance t' of the output entry will satisfy the input-output condition.

Example

In the theory of the real numbers, consider the square-root specification

$$\begin{array}{l} sqrt(r, \epsilon) \Leftarrow \text{find } z \text{ such that} \\ \qquad\qquad z^2 \leq r \ \ \textit{and}\ \ \textit{not}\ \left[(z+\epsilon)^2 \leq r\right] \\ \quad \text{where } 0 \leq r \ \ \textit{and}\ \ 0 < \epsilon \end{array}$$

and the following tableau:

assertions	goals	outputs $sqrt(r, \epsilon)$
1. $0 \leq r$ *and* $0 < \epsilon$		
	2. $z^2 \leq r$ *and* *not* $\left[(z+\epsilon)^2 \leq r\right]$	z
	3. *not* $[\epsilon^2 \leq r]$	0

This tableau is valid in the theory of real numbers, because, under any model of the theory, either the assertion (which has no variables) is false or some instance of one of the two goals is true. (In particular, the instance of goal 2 obtained by taking z to be $\sqrt{r}$ itself is true.)

Under any model for the theory, the output entries of the above tableau are suitable for the square-root specification. In particular, if some instance of goal 2, obtained by replacing z with s, is true, then s will satisfy the input-output condition. That is,

$$\begin{aligned}&\textit{if}\ \ 0 \le r\ \ \textit{and}\ \ 0 < \epsilon\\&\textit{then}\ \ s^2 \le r\ \ \textit{and}\ \ \textit{not}\big[(s+\epsilon)^2 \le r\big]\end{aligned}$$

is true. Also, if assertion 1, which has no output entry, is false, then any term s satisfies the above condition. ⌟

Under a given interpretation $\mathcal{I}$ and for a given specification, two tableaus $\mathcal{T}_1$ and $\mathcal{T}_2$ have the same *meaning* if

$\mathcal{T}_1$ is true under $\mathcal{I}$
if and only if
$\mathcal{T}_2$ is true under $\mathcal{I}$

and

the output entries of $\mathcal{T}_1$ are suitable
if and only if
the output entries of $\mathcal{T}_2$ are suitable.

In a given theory and for a given specification, two tableaus are *equivalent* if, under any model $\mathcal{I}$ for the theory, the meaning of the two tableaus is the same.

PROPERTIES OF A TABLEAU

Let us consider a particular theory and a particular specification, which will both remain fixed throughout this discussion. We shall use the following properties of a tableau:

- *Duality Property*

Any tableau is equivalent to the one obtained by removing an assertion and adding its negation as a new goal, with the same output entry. Similarly, any tableau is equivalent to the one obtained by removing a goal and adding its negation as a new assertion. Thus, we could manage with a system that has no goals or a system that has no assertions, but the distinction between assertions and goals does have some intuitive significance.

- *Renaming Property*

Any tableau is equivalent to the one obtained by systematically renaming the variables of any row. More precisely, we may replace any of the variables of the row with new variables, making sure that all occurrences of the same variable in the row (including those in the output entry) are replaced by the same variable and that distinct variables in the row are replaced by distinct variables. In other words, the variables of a row are dummies that may be renamed freely.

- *Instance Property*

Any tableau is equivalent to the one obtained by introducing as a new row any instance of an existing row. The new row is obtained by replacing all occurrences of certain variables in the existing row (including those in the output entry) with terms. Note that the existing row is not replaced; the new one is simply added.

THE DEDUCTIVE PROCESS

Consider a particular theory and the specification

$$\begin{aligned} f(a) \Leftarrow \text{find } z \text{ such that } & \mathcal{R}[a, z] \\ \text{where } & \mathcal{P}[a]. \end{aligned}$$

We form the initial tableau

assertions	goals	outputs $f(a)$
$\mathcal{P}[a]$		
	$\mathcal{R}[a, z]$	z

We may also include in the initial tableau (as an assertion) any valid sentence of the theory.

Note that the output entries of this tableau are suitable: Under any model for the theory, if the initial assertion $\mathcal{P}[a]$ is false, then any output satisfies the input-output condition vacuously; and if some instance $\mathcal{R}[a, t]$ of the initial goal is true, the corresponding instance t of the associated output entry satisfies the input-output condition. Furthermore, the valid sentences included as initial assertions cannot be false.

We attempt to show that the above tableau is valid. We proceed by applying deduction rules that add new rows without changing the tableau's meaning in any model for the theory. In other words, under a given model, the tableau is true before application of the rule if and only if it is true afterwards, and the output entries are suitable before if and only if they are suitable afterwards. We describe the deduction rules in the next section.

The deductive process continues until we obtain either of the two rows

	true	t

or

false		t

where the output entry t is primitive, i.e., expressed entirely in the vocabulary of our programming language. (We regard the input constant a and the function symbol f as primitive.) At this point, we derive the program

$$f(a) \Leftarrow t.$$

We claim that t satisfies the given specification. For, in applying the deduction rules, we have guaranteed that the new output entries are suitable if the earlier output entries are suitable. We have seen that the initial output entries are all suitable; therefore, the final output entry t is also suitable. This means that, under any model, if the final goal ***true*** is true or the final assertion *false* is false, the corresponding output entry t will satisfy the input-output condition

$$\begin{array}{l} \textit{if} \;\; \mathcal{P}[a] \\ \textit{then} \;\; \mathcal{R}[a,\, t]. \end{array}$$

But under any model the truth symbols *true* and *false* are true and false, respectively, and hence t will satisfy the input-output condition. Therefore, the program $f(a) \Leftarrow t$ does satisfy the specification.

THE DEDUCTION RULES

We now introduce the deduction rules of our system, emphasizing those that play a role in the portions of the square-root derivation we present. We begin with the simplest of the rules.

THE TRANSFORMATION RULES

The transformation rules replace *subexpressions* of an assertion, goal, or output entry with equal or equivalent expressions. For instance, with the transformation rule

$$\mathcal{P} \;\; \textit{and} \;\; \textit{true} \rightarrow \mathcal{P},$$

we can replace the subsentence $((A \;\; \textit{or} \;\; B) \;\; \textit{and} \;\; \textit{true})$ with $(A \;\; \textit{or} \;\; B)$ in the assertion

$((A \;\; \textit{or} \;\; B) \;\; \textit{and} \;\; \textit{true}) \;\; \textit{or} \;\; D$		0

yielding

$(A \;\; \textit{or} \;\; B) \;\; \textit{and} \;\; D$		0

With the transformation rule (in the theory of integers or reals)

$$u + u \rightarrow 2u,$$

we can replace a subterm $(a + b) + (a + b)$ with the term $2(a + b)$.

We use an ***associative-commutative*** matching algorithm (cf. Stickel [81]), so that the associative and commutative properties of operators can be taken into account in applying the transformation rules. Thus, we can use the above rules to replace a subsentence $(\textit{true} \;\; \textit{and} \;\; B)$ with the sentence B and the subterm $(a + b) + b$ with the term $a + 2b$.

We include a complete set of *true-false* transformation rules, such as

> *not false* → *true*
> *if* $\mathcal{P}$ *then false* → *not* $\mathcal{P}$.

Repeated application of these rules can eliminate from a tableau row any occurrence of the truth symbols *true* and *false* as a proper subsentence.

The soundness of the transformation rules is evident, since each produces an expression equivalent or equal (in the theory) to the one to which it is applied.

THE RESOLUTION RULE: GROUND VERSION

The resolution rule corresponds to case analysis in informal reasoning. We first present the *ground version* of the rule, which applies to ground goals. We express it in the following notation:

assertions	goals	outputs $f(a)$
	$\mathcal{F}[\mathcal{P}]$	s
	$\mathcal{G}[\mathcal{P}]$	t

	$\mathcal{F}[true]$
	and
	$\mathcal{G}[false]$
		if $\mathcal{P}$
		then s
		else t

In other words, suppose our tableau contains two ground goals, $\mathcal{F}$ and $\mathcal{G}$, whose output entries are s and t, respectively. Suppose further that $\mathcal{F}$ and $\mathcal{G}$ have a common subsentence $\mathcal{P}$. Then we may derive and add to our tableau the new goal obtained by replacing all occurrences of $\mathcal{P}$ in $\mathcal{F}$ with *true*, replacing all occurrences of $\mathcal{P}$ in $\mathcal{G}$ with *false*, and forming the conjunction of the results. The output entry associated with the derived goal is the conditional expression whose test is the common subexpression $\mathcal{P}$ and whose *then*-clause and *else*-clause are the output entries s and t for $\mathcal{F}$ and $\mathcal{G}$, respectively. Because the resolution rule always introduces occurrences of the truth symbols *true* and *false* as proper subsentences, we can immediately apply *true-false* transformation rules to the derived goal.

For example, suppose our tableau contains the rows

assertions	goals	outputs $f(a,\ b)$
	$\boxed{p(a, b)}$ *and* $q(a)$	a
	not (*if* $r(b)$ *then* $\boxed{p(a,\ b)}$)	b

These goals have a common subsentence $p(a,\ b)$, indicated by boxes. Therefore we may derive and add to our tableau the new goal

	true and $q(a)$
	and
	not (*if* $r(b)$ *then false*)
		if $p(a,\ b)$
		then a
		else b

By repeated application of transformation rules, this goal reduces to

	$q(a)$ *and* $r(b)$	*if* $p(a,\ b)$
		then a
		else b

If one of the given goals has no output entry, the derived output entry is not a conditional expression; it is simply the output entry of the other given goal. If neither given goal has an output entry, the derived goal has no output entry either. We do not require that the two given goals be distinct; we may apply the rule to a goal and itself.

We have presented the resolution rule as it applies to two goals. According to the duality property of tableaus, however, we may transform an assertion into a goal simply by negating it. Therefore, we can apply the rule to an assertion and a goal, or to two assertions.

The resolution rule may be restricted by a *polarity strategy* (Murray [82]; see also Manna and Waldinger [80]), according to which we need not apply the rule unless some occurrence of $\mathcal{P}$ in $\mathcal{F}$ is "positive" and some occurrence of $\mathcal{P}$ in $\mathcal{G}$ is "negative". (Here a subsentence of a tableau is regarded as positive or negative if it is within the scope of a respectively even or odd number of negation connectives. Each assertion is considered to be within the scope of an implicit negation; thus, while goals are positive, assertions are negative. The *if*-clause $\mathcal{P}$ of a subsentence (*if* $\mathcal{P}$ *then* $\mathcal{Q}$) is considered to be within the scope of an additional implicit negation.) This strategy allows us to disregard many useless applications of the rule.

Let us show that the *resolution rule is sound*; that is, in a given model of the theory and for a given specification, the meaning of the tableau is the same before and after application of the rule. It actually suffices to show that, if the derived goal is true, then at least one of the given goals is true; and if the given output entries are suitable, so is the derived output entry.

Suppose the derived goal ($\mathcal{F}[$*true*$]$ *and* $\mathcal{G}[$*false*$]$) is true. Then both its conjuncts $\mathcal{F}[$*true*$]$ and $\mathcal{G}[$*false*$]$ are true. We distinguish between two cases, depending on whether or not the common subsentence $\mathcal{P}$ is true or false. In the case in which $\mathcal{P}$ is true, the [ground] goal $\mathcal{F}[\mathcal{P}]$ has the same truth-value as the conjunct $\mathcal{F}[$*true*$]$; that is, $\mathcal{F}[\mathcal{P}]$ is true. In the case in which $\mathcal{P}$ is false, the goal $\mathcal{G}[\mathcal{P}]$ has the same truth-value as the conjunct $\mathcal{G}[$*false*$]$; that is, $\mathcal{G}[\mathcal{P}]$ is true. In either case, one of the two given goals, $\mathcal{F}[\mathcal{P}]$ and $\mathcal{G}[\mathcal{P}]$, is true.

Now assume that the given output entries are suitable. To show that the derived output entry is suitable, we suppose that the derived goal is true and establish that the derived output entry satisfies the input-output condition. We have seen that, in the case in which $\mathcal{P}$ is true, the given goal $\mathcal{F}[\mathcal{P}]$ is true; because its output entry s is suitable, it satisfies the input-output condition. Similarly, in the case in which $\mathcal{P}$ is false, the term t satisfies the input-output condition. In either case, therefore, the conditional expression (*if* $\mathcal{P}$ *then* s *else* t) satisfies the input-output condition; but this is the derived output entry.

THE RESOLUTION RULE: GENERAL VERSION

We have described the ground version of the resolution rule, which applies to goals with no variables. We now present the general version, which applies to goals with variables. In this case, we can apply a substitution to the goals, as necessary, to create a *common* subsentence.

assertions	goals	outputs $f(a)$
	$\mathcal{F}[\mathcal{P}]$	s
	$\mathcal{G}[\hat{\mathcal{P}}]$	t

	$\mathcal{F}\theta[true]$
	and
	$\mathcal{G}\theta[false]$
		if $\mathcal{P}\theta$
		then $s\theta$
		else $t\theta$

More precisely, suppose our tableau contains goals $\mathcal{F}$ and $\mathcal{G}$, which have no variables in common. (This can be ensured by renaming the variables of the rows as necessary, according to the *renaming* property.) Suppose further that some of the subsentences of $\mathcal{F}$ and some of the subsentences of $\mathcal{G}$ are unifiable, with a most-general unifier θ; let $\mathcal{P}\theta$ be the unified subsentence. Then we may derive and add to our tableau the new goal obtained by replacing all occurrences of $\mathcal{P}\theta$ in $\mathcal{F}\theta$ with ***true***, replacing all occurrences of $\mathcal{P}\theta$ in $\mathcal{G}\theta$ with *false*, and forming the conjunction of the results. The associated output entry is a conditional expression whose test is the unified subsentence $\mathcal{P}\theta$ and whose *then*-clause and *else*-clause are the corresponding instances $s\theta$ and $t\theta$, respectively, of the given output entries.

In other words, to apply the general version of the rule to $\mathcal{F}$ and $\mathcal{G}$, we apply the ground version of the rule to $\mathcal{F}\theta$ and $\mathcal{G}\theta$. The soundness of the general version follows from the soundness of the ground version. The polarity strategy applies as before. If we wish to apply the rule to an assertion and a goal or to two assertions, we can regard the assertions as goals by negating them, as in the ground case.

For example, suppose our tableau contains the rows

assertions	goals	outputs $f(a,\ b)$
	$\boxed{\begin{array}{l} y \le a \ \ and \\ not\ [y + b \le a] \end{array}}$ *and* $p(y)$	$g(y)$
if $q(x,\ v)$ *then* $\boxed{\begin{array}{l} f(x,\ v) \le x \ \ and \\ not\ [f(x,\ v) + v \le x] \end{array}}$		

The boxed subsentences are unifiable; a most-general unifier is

$$\theta : \quad \{x \leftarrow a,\ v \leftarrow b,\ y \leftarrow f(a,\ b)\}.$$

The subsentences are respectively positive and negative, as indicated by the annotation. We may regard the assertion as a goal by negating it. By application of the general version of the resolution rule, we may derive the new row

	$\begin{bmatrix} \mathit{true}\ \ \mathit{and} \\ p(f(a,\ b)) \end{bmatrix}$ and $\mathit{not} \begin{bmatrix} \mathit{if}\ \ q(a,\ b) \\ \mathit{then}\ \ \mathit{false} \end{bmatrix}$	$g(f(a,\ b))$

By the application of *true-false* transformation rules, this goal reduces to

	$p(f(a,\ b))$ and $q(a,\ b)$	$g(f(a,\ b))$

Note that the unifier θ has been applied to all variables in the given rows, including those in the output entry. Because the given assertion has no output entry, the derived output entry is not a conditional expression. This application of the rule is in accordance with the polarity strategy.

The resolution rule and the *true-false* transformation rules have been shown by Murray [82] to constitute a complete system for first-order logic. The polarity strategy maintains this completeness.

We use an associative-commutative unification algorithm (as in Stickel [81]) so that the associative and commutative properties of such operators as addition and conjunction can be taken into account in finding a unifier; thus, $p\big(f(x) + (b + g(a))\big)$ can be unified with $p\big((g(y) + f(b)) + x\big)$.

We have introduced two additional rules to give special treatment to equality and other important relations (Manna and Waldinger [85]), but these rules play no part in the portion of the derivation to be discussed.

We shall need the induction rule; this we describe next.

THE MATHEMATICAL INDUCTION RULE

The rules presented so far do not allow us to introduce any repetitive construct into the program being derived. The induction rule accounts for the introduction of recursion in the derived program. We employ a single well-founded induction rule, which applies to a variety of theories.

A well-founded relation $\prec_w$ is one that admits no infinite decreasing sequences, i.e., sequences $x_1, x_2, x_3, \ldots$, such that

$$x_1 \succ_w x_2 \ \ \mathit{and}\ \ x_2 \succ_w x_3 \ \ \mathit{and} \ldots.$$

For instance, the less-than relation $<$ is well-founded in the theory of nonnegative integers, but not in the theory of real numbers.

The version of the well-founded induction rule we need for the derivation is expressed as follows (the general version is more complex):

Suppose our initial tableau is

assertions	goals	outputs $f(a)$
$\mathcal{P}[a]$		
	$\mathcal{R}[a,\ z]$	z

In other words, we are attempting to construct a program f that, for an arbitrary input a, yields an output z satisfying the input-output condition

$$\begin{array}{l} \textit{if}\ \ \mathcal{P}[a] \\ \textit{then}\ \ \mathcal{R}[a,\ z]. \end{array}$$

According to the well-founded induction rule, we may prove this assuming as our induction hypothesis that the program f will yield an output $f(x)$ satisfying the same input-output condition

$$\begin{array}{l} \textit{if}\ \ \mathcal{P}[x] \\ \textit{then}\ \ \mathcal{R}[x,\ f(x)], \end{array}$$

provided that x is less than a with respect to some well-founded relation $\prec_w$, that is, $x \prec_w a$. In other words, we may add to our tableau the new assertion

$\textit{if}\ \ x \prec_w a$ $\textit{then}\ \ \textit{if}\ \ \mathcal{P}[x]$ $\qquad \textit{then}\ \ \mathcal{R}[x,\ f(x)]$		

The well-founded relation $\prec_w$ used in the induction rule is arbitrary and must be selected later in the proof.

For example, consider the initial tableau obtained from the square-root specification:

assertions	goals	outputs $sqrt(r,\ \epsilon)$
$0 \le r\ \ \textit{and}\ \ 0 < \epsilon$		
	$z^2 \le r\ \ \textit{and}$ $\textit{not}\ \left[(z+\epsilon)^2 \le r\right]$	z

By application of the well-founded induction rule, we may introduce as a new assertion the induction hypothesis

$\textit{if}\ \ \langle x,\ v\rangle \prec_w \langle r,\ \epsilon\rangle$ $\textit{then}\ \ \textit{if}\ \ 0 \le x\ \ \textit{and}\ \ 0 < v$ $\qquad \textit{then}\ \ \left[\begin{array}{l} \left[sqrt(x,\ v)\right]^2 \le x\ \ \textit{and} \\ \textit{not}\ \left(\left[sqrt(x,\ v)+v\right]^2 \le x\right) \end{array}\right]$		

In other words, we may assume inductively that the output of the square-root program we construct will satisfy the input-output condition for inputs x and υ that are less than the given inputs r and ϵ with respect to some well-founded relation $\prec_w$.

Use of the induction hypothesis in the proof may account for the introduction of a recursive call into the derived program. For example, suppose that in the square-root derivation we manage to develop a goal of form

	$\mathcal{G}\left[\begin{array}{l} z^2 \le s \ \ and \\ not\ \boxed{(z+\delta)^2 \le s} \end{array}\right]$	$t[z]$

The boxed subsentences of this goal and the induction hypothesis are unifiable; a most-general unifier is

$$\theta = \{x \leftarrow s,\ \upsilon \leftarrow \delta,\ z \leftarrow sqrt(s,\ \delta)\}.$$

Therefore we can apply the resolution rule to obtain the new goal

	$\mathcal{G}[true]$ and $not\ \left[\begin{array}{l} if\ \langle s,\ \delta\rangle \prec_w \langle r,\ \epsilon\rangle \\ then\ \ if\ 0 \le s\ \ and\ \ 0 < \delta \\ \qquad then\ \ false \end{array}\right]$	$t[sqrt(s,\ \delta)]$

This goal reduces under transformation to

	$\mathcal{G}[true]$ and $\langle s,\ \delta\rangle \prec_w \langle r,\ \epsilon\rangle$ and $0 \le s$ and $0 < \delta$	$t[sqrt(s,\ \delta)]$

Note that a recursive call $sqrt(s,\ \delta)$ has been introduced into the output entry as a result of this step. The condition ($0 \le s$ *and* $0 < \delta$) in the goal ensures the legality of the arguments s and δ, i.e., that they satisfy the input condition of the desired program. The condition $\langle s,\ \delta\rangle \prec_w \langle r,\ \epsilon\rangle$ ensures that the evaluation of the recursive call cannot lead to a nonterminating computation. (If there were an infinite computation, we could construct a corresponding infinite sequence of pairs of arguments decreasing with respect to $\prec_w$, thus contradicting the definition of a well-founded relation.)

The particular well-founded relation $\prec_w$ referred to in the induction hypothesis is not yet specified; it is selected at a later stage of the proof. If we allow well-founded relations to be objects in our domain, we may regard the sentence $x \prec_w y$ as an abbreviation for $\prec(w,\ x,\ y)$; thus, w is a variable that may be instantiated to a particular relation. We assume that the properties of many known well-founded relations (such as $\prec_{tree}$, the proper-subtree relation over trees) and of functions for combining them are among the assertions of our intial tableau.

We have given the simplest version of the induction rule, which is applied only to the initial rows of the tableau; in its general version, we may apply the rule to any of the rows, and we may strengthen or generalize the rows to which the rule is applied. In this more general version, the rule accounts for the introduction of auxiliary subprograms into the program being constructed. We shall avoid discussion of auxiliary subprograms here.

We are now ready to present the most interesting segment of the derivation of the square-root program.

THE DERIVATION

Recall that, in the theory of real numbers, the specification for the real-number square-root program is

$$\begin{aligned} &sqrt(r, \epsilon) \Leftarrow \text{find } z \text{ such that} \\ &\qquad\qquad z^2 \le r \;\; and \;\; not \left[(z + \epsilon)^2 \le r\right], \\ &\quad \text{where } 0 \le r \;\; and \;\; 0 < \epsilon . \end{aligned}$$

In other words, we want to find an estimate z that is within a tolerance ϵ less than $\sqrt{r}$, the exact square root of r, where we may assume that r is nonnegative and ϵ is positive.

We begin accordingly with the tableau

assertions	goals	outputs $sqrt(r, \epsilon)$
1. $0 \le r \;\; and \;\; 0 < \epsilon$		
	2. $\boxed{z^2 \le r}^{+} \;\; and \;\; not \left[(z + \epsilon)^2 \le r\right]$	z

The assertion and goal of this tableau are the input and output conditions, respectively, of the given specification; the output entry of the goal is the output variable of the program.

THE DISCOVERY OF BINARY SEARCH

We are about to apply the resolution rule to goal 2 and itself. To make this step easier to understand, let us write another copy of goal 2.

	2'. $\hat{z}^2 \le r \;\; and \;\; not \boxed{(\hat{z} + \epsilon)^2 \le r}$	$\hat{z}$

We have renamed the variable of the second copy of the goal, so that the two copies have no variables in common.

The boxed subsentences of the two copies of the goal are unifiable; a most-general unifier is

$$\theta : \quad \{z \leftarrow \hat{z} + \epsilon\}.$$

Therefore, we can apply the resolution rule between the two copies of goal 2 to obtain

	true and not $\left[((\hat{z}+\epsilon)+\epsilon)^2 \le r\right]$
	and
	$\hat{z}^2 \le r$ *and not false*
		if $(\hat{z}+\epsilon)^2 \le r$
		then $\hat{z}+\epsilon$
		else $\hat{z}$

By application of transformation rules, including the rule

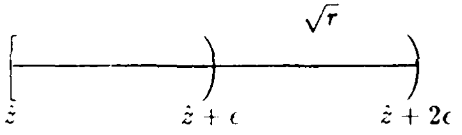
$$u + u \rightarrow 2u,$$

this goal can be reduced to

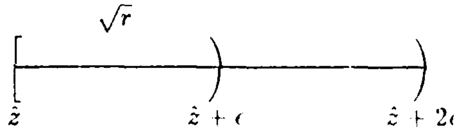
	3. $\boxed{\begin{array}{l} \hat{z}^2 \le r \\ \quad \textit{and} \\ \textit{not } \left[(\hat{z}+2\epsilon)^2 \le r\right] \end{array}}$	*if* $(\hat{z}+\epsilon)^2 \le r$
		then $\hat{z}+\epsilon$
		else $\hat{z}$

(We have reordered the conjuncts for pedagogical reasons only; because we use associative-commutative unification, their actual order is irrelevant.)

According to goal 3, it suffices to find a rougher estimate $\hat{z}$, which is within a tolerance 2ϵ less than $\sqrt{r}$, the exact square root of r. For then either $\hat{z}+\epsilon$ or $\hat{z}$ itself will be within ϵ less than $\sqrt{r}$, depending on whether or not $\hat{z}+\epsilon$ is less than or equal to $\sqrt{r}$. The two possibilities are illustrated below:



Case: $\hat{z}+\epsilon \le \sqrt{r}$



Case: *not* $[\hat{z}+\epsilon \le \sqrt{r}\,]$

Goal 3 contains the essential idea of binary search as applied to the square-root problem. Although the idea seems subtle to us, it appears almost immediately in the derivation. The step is nearly inevitable: any brute-force search procedure would discover it.

The derivation of goal 3 is logically straightforward, but the intuition behind it may be a bit mysterious. Let us paraphrase the reasoning in a more geometric way. Our initial goal 2 expresses that it suffices to find a real number z such that $\sqrt{r}$ belongs to the half-open interval $[z,\ z+\epsilon)$. Our rewritten goal 2′ expresses that it is equally acceptable to find a real number $\hat{z}$ such that $\sqrt{r}$ belongs to the half-open interval $[\hat{z},\ \hat{z}+\epsilon)$. We shall be content to achieve either of these goals; i.e., we shall be happy if $\sqrt{r}$ belongs to either of the two half-open intervals. In taking $\hat{z}$ to be $z+\epsilon$, we are concatenating the two intervals, obtaining a new half-open interval $[z,\ z+2\epsilon)$ twice the length of the original. It suffices to find a real number z such that $\sqrt{r}$ belongs to this new, longer interval, because then $\sqrt{r}$ must belong to one or the other of the two smaller ones.

INTRODUCTION OF THE RECURSIVE CALLS

Let us continue the derivation one more step. By the well-founded induction rule, we may introduce the induction hypothesis

$\mathit{if}\ \langle x,\ v\rangle \prec_w \langle r,\ \epsilon\rangle$ $\mathit{then}\ \mathit{if}\ 0 \le x\ \mathit{and}\ 0 < v$ $\mathit{then}\ \boxed{\begin{array}{l}[\mathit{sqrt}(x,\ v)]^2 \le x\ \mathit{and}\\ \mathit{not}\ ([\mathit{sqrt}(x,\ v) + v]^2 \le x)\end{array}}.$		

In other words, we assume inductively that the output $\mathit{sqrt}(x,\ v)$ of the program will satisfy the input-output condition for any inputs x and v such that $\langle x,\ v\rangle \prec_w \langle r,\ \epsilon\rangle$. The boxed subsentences of goal 3 and the induction hypothesis are unifiable; a most-general unifier is

$$\theta :\quad \{x \leftarrow r,\ v \leftarrow 2\epsilon,\ \hat{z} \leftarrow \mathit{sqrt}(r,\ 2\epsilon)\}.$$

We obtain (after *true-false* transformation)

	4. $\langle r,\ 2\epsilon\rangle \prec_w \langle r,\ \epsilon\rangle$ and $0 \le r\ \ \mathit{and}\ 0 < 2\epsilon$	$\mathit{if}\ [\mathit{sqrt}(r,\ 2\epsilon) + \epsilon]^2 \le r$ $\mathit{then}\ \mathit{sqrt}(r,\ 2\epsilon) + \epsilon$ $\mathit{else}\ \mathit{sqrt}(r,\ 2\epsilon)$

Note that at this point three recursive calls $\mathit{sqrt}(r,\ 2\epsilon)$ have been introduced into the output entry. The condition ($0 \le r$ *and* $0 < 2\epsilon$) ensures that the arguments r and 2ϵ of these recursive calls will satisfy the input condition for the program, that r is nonnegative and 2ϵ is positive. The condition $\langle r,\ 2\epsilon\rangle \prec_w \langle r,\ \epsilon\rangle$ ensures that the newly introduced recursive calls cannot lead to a nonterminating computation. The well-founded relation $\prec_w$ that serves as the basis for the induction is as yet unspecified.

We omit those portions of the derivation that account for the introduction of the base case and the choice of the well-founded relation. The final program we obtain is

```
sqrt(r, ε)  ⇐  if ε ≤ max(r, 1)
               then if [sqrt(r, 2ε) + ε]² ≤ r
                    then sqrt(r, 2ε) + ε
                    else sqrt(r, 2ε)
               else 0.
```

A few words on this program are in order.

DISCUSSION OF THE PROGRAM

The program first checks whether the error tolerance ϵ is reasonably small. If ϵ is very big, that is, if $\mathit{max}(r,\ 1) < \epsilon$, then the output can safely be taken to be 0. For, because $0 \le r$, we have

$$0^2 \le r.$$

And because $\mathit{max}(r,\ 1) < \epsilon$, we have $r < \epsilon$ and $1 < \epsilon$, and hence $r < \epsilon^2$ - that is,

$$\mathit{not}\ [(0 + \epsilon)^2 \le r].$$

Thus [illegible] output condition in this case.

[illegible] the program finds a rougher estimate $sqrt(r,\ 2\epsilon)$, which is within 2ϵ [illegible] asks whether increasing this estimate by ϵ will leave it less than [illegible] increased by ϵ; if not, the rough estimate is already close enough.

The termination of this program is a bit problematic, because the argument ϵ is doubled with each recursive call. However, the argument r is unchanged and recursive calls are evaluated only in the case [illegible] upper bound on these increasing arguments. More precisely, the well-founded relation [illegible] selected in the proof is one such that

$$\langle r,\ 2y\rangle \prec [illegible]\ y\rangle$$

provided that $0 < y \leq max(r,\ 1)$

If the multiple occurrences of the recursive call $sqrt(r,\ 2\epsilon)$ are combined by eliminating common subexpressions, the program we obtain is reasonably efficient; it requires $\lceil log_2(max(r,\ 1)/\epsilon)\rceil$ recursive calls.

Our final program is somewhat different from the iterative program we considered in the beginning. The iterative program divides an interval in half at each iteration; the recursive program doubles an interval with each recursive call. Division of the interval in half occurs implicitly as the recursive program unwinds, i.e., when the recursive calls yield output values.

It is possible to obtain a version of the iterative program by formal derivation within the deductive-tableau system. Although the derivation and the resulting program are more complex (it requires two additional inputs), it was this derivation we discovered first, because we were already familiar with the iterative program.

We first found the recursive program in examining the consequences of purely formal derivation steps, not because we expected them to lead to a program but because we were looking for strategic considerations that would rule them out. When we examined the program initially, we suspected an error in the derivation. We had not seen programs of this form before, and we certainly would not have constructed this one by informal means.

ANALOGOUS ALGORITHMS

Many binary-search algorithms have been derived in an analogous way. Let us first consider some other real-numerical problems.

REAL-NUMBER ALGORITHMS

Suppose a program to perform real-number division is specified as follows:

$$\begin{aligned} div(r,\ s,\ \epsilon) \ \Leftarrow\ & \text{find } z \text{ such that} \\ & z \cdot s \leq r \ \ and\ \ not\ \big[(z+\epsilon)\cdot s \leq r\big] \\ \text{where } 0 \leq r \ \ and\ \ 0 < s \ \ and\ \ 0 < \epsilon. & \end{aligned}$$

In other words, the program is required to yield a real number z that is within a tolerance ϵ less than r/s, the exact quotient of dividing r by s. We obtain the program

```
$div(r,\ s,\ \epsilon)$  $\Leftarrow$  $if\ \epsilon \cdot s \le r$
                      $then\ if\ \left[div(r,\ s,\ 2\epsilon) + \epsilon\right] \cdot s \le r$
                              $then\ div(r,\ s,\ 2\epsilon) + \epsilon$
                              $else\ div(r,\ s,\ 2\epsilon)$
                      $else\ 0$.
```

The rationale for this program, like its derivation, is analogous to that for the real-number square root. The program first checks whether the error tolerance is reasonably small, that is, if $\epsilon \cdot s \le r$. If ϵ is very big, that is, if $r < \epsilon \cdot s$, then the output can be taken safely to be 0. For because $0 \le r$, we have

$$0 \cdot s \le r.$$

And because $r < \epsilon \cdot s$, we have $r < (0 + \epsilon) \cdot s$, that is,

$$not\ \left[(0 + \epsilon) \cdot s \le r\right].$$

Thus, 0 satisfies both conjuncts of the output condition in this case.

On the other hand, if ϵ is small, that is, if $\epsilon \cdot s \le r$, the program finds a rougher estimate $div(r,\ s,\ 2\epsilon)$, which is within 2ϵ less than r/s. The program considers whether increasing this estimate by ϵ will leave it less than r/s. If so, the rough estimate may be increased by ϵ; if not, the rough estimate is already close enough.

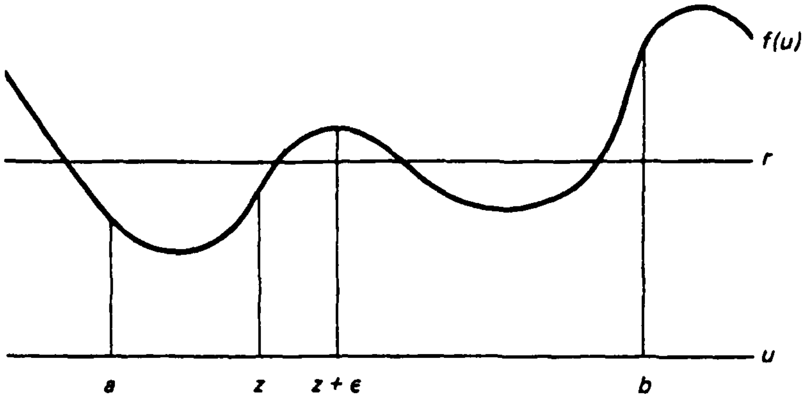
The termination proof for this program is also analogous to that for the square root. Although the argument ϵ is doubled with each recursive call, the other arguments are unchanged and the calls are evaluated only in the case in which $\epsilon \cdot s \le r$, that is, $\epsilon \le r/s$. Thus, there is a uniform upper bound on the doubled argument.

It may be clear from the above discussion that there is little in the derivations for the square-root and division programs that depends on the properties of these functions. More or less the same derivation suffices to find an approximate solution to an arbitrary real-number equation $f(z) = r$.

For a given computable function f, we consider the specification

$$\begin{aligned} solve(r,\ \epsilon) \Leftarrow\ & \text{find } z \text{ such that} \\ & \quad f(z) \le r \ \ and\ \ not\ \left[f(z + \epsilon) \le r\right] \\ \text{where}\ \ & f(a) \le r \ \ and\ \ \begin{bmatrix} if\ b < u \\ then\ \ not\ \left(f(u) \le r\right) \end{bmatrix}. \end{aligned}$$

Here a and b are primitive constants and u is a variable. In other words, we assume that there exist real numbers a and b such that $f(a) \le r$ and $f(u) > r$ for every real u greater than b. The specification is illustrated as follows:



Note that we do not need to assume f is increasing or even continuous; if f is not continuous, an exact solution to the equation $f(a) = r$ need not exist, but only an approximate solution is required by the specification.

The program we obtain is

```
$solve(r,\ \epsilon)$  ⇐  if $a + \epsilon \le b$
                          then if $f(solve(r,\ 2\epsilon) + \epsilon) \le r$
                                then $solve(r,\ 2\epsilon) + \epsilon$
                                else $solve(r,\ 2\epsilon)$
                          else $a$.
```

In the recursive case, in which $a + \epsilon \le b$, the program is so closely analogous to the previous binary-search programs as to require no further explanation. In the base case, in which $b < a + \epsilon$, the output can safely be taken to be a. For, by our input condition, we have

$$f(a) \le r$$

and (again by our input condition, because $b < a + \epsilon$)

$$not\ [f(a + \epsilon) \le r].$$

Thus, a satisfies both conjuncts of the output condition in this case.

The above program may be regarded as a schema, because we may take the symbol f to be any primitive function symbol. An even more general binary-search program schema can be derived from the specification

$$search(r,\ \epsilon) \Leftarrow \text{find } z \text{ such that } p(r,\ z)\ \ and\ \ not\, p(r,\ z + \epsilon)$$

$$\text{where}\ \ p(a)\ \ and\ \ \begin{bmatrix} if\ \ b < u \\ then\ \ not\, p(r, u) \end{bmatrix},$$

where p is a primitive relation symbol and a and b are primitive constants. We obtain the schema

```
$search(r,\ \epsilon)$  ⇐  if $a + \epsilon \le b$
                           then if $p(r, search(r,\ 2\epsilon) + \epsilon)$
                                 then $search(r,\ 2\epsilon) + \epsilon$
                                 else $search(r,\ 2\epsilon)$
                           else $a$.
```

INTEGER ALGORITHMS

The programs we have discussed apply to the nonnegative real numbers; using the same approach, we have derived analogous programs that apply to the nonnegative integers. These derivations require a generalization step in applying the induction rule. We have avoided presenting generalization and the concomitant introduction of auxiliary programs in this paper, but we give some results of these derivations here.

Integer square root

The integer square-root program is intended to find the integer part of $\sqrt{n}$, the real square root of a nonnegative integer n. It can be specified in the theory of nonnegative integers as follows:

```
$sqrt(n)$  $\Leftarrow$  find $z$ such that
                   $z^2 \leq n$ and not$[(z+1)^2 \leq n]$.
```

In other words, the program must yield a nonnegative integer z that is within 1 less than $\sqrt{n}$.

In the course of the derivation, we are led to introduce an auxiliary program to meet the more general specification

```
$sqrt2(n, i)$  $\Leftarrow$  find $z$ such that
                       $z^2 \leq n$ and not $[(z+i)^2 \leq n]$
    where $0 < i$.
```

In other words, we wish to find a nonnegative integer z that is within i less than $\sqrt{n}$. This auxiliary specification is precisely analogous to the real-number square-root specification, with i playing the role of the error tolerance ϵ.

The programs we obtain to meet these specifications are

```
$sqrt(n)$  $\Leftarrow$  $sqrt2(n, 1)$,
```

where

```
$sqrt2(n, i)$  $\Leftarrow$  if $i \leq n$
                   then if $[sqrt2(n, 2i) + i]^2 \leq n$
                        then $sqrt2(n, 2i) + i$
                        else $sqrt2(n, 2i)$
                   else $0$.
```

Integer quotient

The integer quotient program can be specified similarly:

```
$quot(m, n)$  $\Leftarrow$  find $z$ such that
                      $z \cdot n \leq m$ and not $[(z+1) \cdot n \leq m]$
    where $0 < n$.
```

In other words, we wish to find a nonnegative integer z that is within 1 less than m/n, the real-number quotient of m and n.

In the course of the derivation, we are led to introduce an auxiliary program to meet the more general specification

$$\begin{array}{lll} quot3(m,\ n,\ i) & \Leftarrow & \text{find } z \text{ such that} \\ & & \qquad z \cdot n \leq m \ \ \mathit{and}\ \ not\big[(z+i)\cdot n \leq m\big] \\ \quad \text{where } 0 < n \ \ \mathit{and} \ \ 0 < i. & & \end{array}$$

In other words, we wish to find a nonnegative integer z that is within i less than m/n.

The programs obtained to meet these specifications are

$$quot(m,\ n) \ \Leftarrow\ quot3(m,\ n,\ 1)$$

where

```
$quot3(m, n, i)$  $\Leftarrow$  $\mathit{if}\ i \cdot n \leq m$
                     $\mathit{then}\ \mathit{if}\ \big[quot3(m, n, 2i) + i\big] \cdot n \leq m$
                          $\mathit{then}\ quot3(m, n, 2i) + i$
                          $\mathit{else}\ quot3(m, n, 2i)$
                     $\mathit{else}\ 0.$
```

The derivation is again analogous.

DISCUSSION

The derivations were first discovered manually: the real-number square-root derivation was subsequently reproduced by Yellin in an interactive program-synthesis system. The only automatic implementation of the system (Russell [83]) is unable to construct the derivation for a simple reason: it never attempts to apply the resolution rule to a goal and itself.

The results of this investigation run counter to our usual experience. It is common for a bit of reasoning that seems simple and intuitively straightforward to turn out to be difficult to formalize and more difficult still to duplicate automatically. Here the opposite is true: an idea that requires a substantial leap of human ingenuity to discover is captured mechanically in a few easy formal steps.

ACKNOWLEDGMENTS

We would like to thank Martin Abadi, Yoni Malachi, Eric Muller, Mark Stickel, Jonathan Traugott, and Frank Yellin for discussions and helpful suggestions on the subject of this paper.

REFERENCES

Dershowitz and Manna [77]

N. Dershowitz and Z. Manna, The evolution of programs: Automatic program modification, *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 6, November 1977, pp. 377 385.

Manna and Waldinger [80]

Z. Manna and R. Waldinger, A deductive approach to program synthesis, *ACM Transactions on Programming Languages and Systems*, Vol. 2, No. 1, January 1980, pp. 90 121.

Manna and Waldinger [85]

Z. Manna and R. Waldinger, Special relations in automated deduction, *Journal of the ACM*, 1985, to appear.

Murray [82]

N. V. Murray, Completely nonclausal theorem proving, *Artificial Intelligence*, Vol. 18, No. 1, 1982, pp. 67-85.

Robinson [79]

J. A. Robinson, *Logic: Form and Function*, North-Holland, New York, N. Y., 1979.

Russell [83]

S. Russell, PSEUDS: A programming system using deductive synthesis, Technical Report, Computer Science Department, Stanford University, Stanford, Calif., September 1983.

Smith [85]

D. R. Smith, Top-down synthesis of simple divide-and-conquer algorithms, *Artificial Intelligence*, 1985, to appear.

Stickel [81]

M. E. Stickel, A unification algorithm for associative-commutative functions, *Journal of the ACM*, Vol. 28, No. 3, July 1981, pp. 423 434.

Wensley [59]

J. H. Wensley, A class of nonanalytical iterative processes, *Computer Journal*, Vol. 1, January 1959, pp. 163-167.

May 1985

Report No. STAN-CS-85-1051

Special Relations in Automated Deduction

by

Zohar Manna
Richard Waldinger

Department of Computer Science

Stanford University
Stanford, CA 94305

SPECIAL RELATIONS IN AUTOMATED DEDUCTION

Zohar Manna
Computer Science Department
Stanford University

Richard Waldinger
Artificial Intelligence Center
SRI International

ABSTRACT

Two deduction rules are introduced to give streamlined treatment to relations of special importance in an automated theorem-proving system. These rules, the *relation replacement* and *relation matching* rules, generalize to an arbitrary binary relation the paramodulation and E-resolution rules, respectively, for equality, and may operate within a nonclausal or clausal system. The new rules depend on an extension of the notion of *polarity* to apply to subterms as well as to subsentences, with respect to a given binary relation. The rules allow us to eliminate troublesome axioms, such as transitivity and monotonicity, from the system; proofs are shorter and more comprehensible, and the search space is correspondingly deflated.

1. INTRODUCTION

In any theorem-proving system, the task of representing properties of objects is shared between axioms and rules of inference. The axioms of the system are easier to introduce and modify, because they are expressed in a logical language. However, because axioms are declarative rather than imperative, they are given no individual heuristic controls. The rules of inference, on the other hand, cannot be altered without reprogramming the system, and they are usually expressed in the system's programming language. However, the rules can be given individual heuristic controls and strategies.

It is customary to use rules of inference to express properties of the logical connectives, which are the same from one theory to the next, and to use axioms to express properties of constants, functions, and relations, which may vary. It is hazardous, however, to express certain properties of functions and relations by axioms. Some properties of the equality relation, for example, are rarely represented axiomatically. For one thing, in a first-order system indefinitely many axioms are necessary to represent the substitutivity property of this relation, depending on how many function and relation symbols are in the vocabulary of the theory.

For instance, for a binary function symbol $f(x, y)$, we must introduce two *functional-substitutivity* axioms,

$$\begin{array}{l} \textit{if } x = y \\ \textit{then } f(x, z) = f(y, z) \end{array} \quad \text{and} \quad \begin{array}{l} \textit{if } x = y \\ \textit{then } f(z, x) = f(z, y), \end{array}$$

and for a binary predicate symbol $p(x, y)$, we must introduce two *predicate-substitutivity* axioms,

$$\begin{array}{l} \textit{if } x = y \\ \textit{then if } p(x, z) \textit{ then } p(y, z) \end{array} \quad \text{and} \quad \begin{array}{l} \textit{if } x = y \\ \textit{then if } p(z, x) \textit{ then } p(z, y). \end{array}$$

An abbreviated version of this paper appears in the proceedings of the Twelfth International Colloquium on Automata, Languages, and Programming (ICALP), Nafplion, Greece, July 1985.

This research was supported in part by the National Science Foundation under grants MCS-82-14523 and MCS-81-05565, by the Defense Advanced Research Projects Agency under contract N00039-84-C-0211, by the United States Air Force Office of Scientific Research under contract AFOSR-81-0014, by the Office of Naval Research under contract N00014-84-C-0706, and by a contract from the International Business Machines Corporation.

(We tacitly quantify variables universally over the entire sentence.) In general, for each n-ary function symbol $f(x_1, \ldots, x_n)$, we introduce n *functional-sustitutivity* axioms. Similarly, for each n-ary predicate symbol $p(x_1, \ldots, x_n)$, we introduce n *predicate-substitutivity* axioms.

More importantly, axioms for equality are difficult to control strategically, because they have many irrelevant consequences. An axiom such as *transitivity*,

$$\begin{array}{l} \textit{if } x = y \textit{ and } y = z \\ \textit{then } x = z, \end{array}$$

will allow us to derive logical consequences from any sentence mentioning the equality relation. Few of these consequences will have any bearing on the proof.

In response to this problem, some theorem-proving researchers have paraphrased their theories to avoid explicit mention of the equality axiom (e.g., Kowalski [79]). Others have adopted special inference rules for dealing with equality. In resolution systems, two equality rules, paramodulation (Wos and Robinson [69]) and E-resolution (Morris [69]) have been found to be effective. Variations of these rules are used in many theorem provers today (e.g., Boyer and Moore [79], Digricoli [83]). By a single application of either of these rules, we can derive conclusions that would require several steps if the properties of equality were represented axiomatically. The proofs are markedly shorter, and the search spaces are even more dramatically compressed because the axioms and intermediate steps are not required. Within their limited domain of application, theorem-proving systems using these rules surpass most human beings in their capabilities.

SPECIAL RELATIONS

The authors became involved in theorem proving because of its application to program synthesis, the derivation of a program to meet a given specification. We have been pursuing a deductive approach to this problem, under which computer programming is regarded as a theorem-proving task. In the proofs required for program synthesis, certain relations assume special importance. Again and again, proofs require us to reason not only about the equality relation, but also about the less-than relation $<$ (over the integers or reals), the subset relation $\subseteq$, the sublist relation $\preceq_{list}$, or the subtree relation $\preceq_{tree}$. To represent the transitivity and other properties of these relations axiomatically leads to many of the same problems that were faced in dealing with equality: the axioms apply almost everywhere, spawning innumerable consequences that swamp the system. Yet we would not want to implement a new inference rule for each of the relations we find important.

Both the paramodulation and the E-resolution rules are based on the *substitutivity* property of equality, that if two elements are equal they may be used interchangeably; i.e., for any sentence $\mathcal{P}\langle x, y\rangle$, the sentence

$$\begin{array}{l} \textit{if } x = y \\ \textit{then if } \mathcal{P}\langle x, y\rangle \textit{ then } \mathcal{P}\langle y, x\rangle \end{array}$$

is valid. Here $\mathcal{P}\langle y, x\rangle$ is the result of replacing in $\mathcal{P}\langle x, y\rangle$ certain (perhaps none) of the occurrences of x with y, and certain (perhaps none) of the occurrences of y with x. (The notations we use here informally will be defined systematically later on. We assume throughout that sentences are quantifier-free.)

We observe that many of the relations we regard as important exhibit substitutivity properties similar to the above property of equality, but under restricted circumstances. For example, over the nonnegative integers, we can show that

$$\begin{array}{l} \textit{if } x \leq y \\ \textit{then if } a \leq x \cdot b \\ \qquad \textit{then } a \leq y \cdot b \end{array}$$

and, over the lists, we can show that

$$\begin{array}{l} \mathit{if}\ x \preceq_{list} y \\ \mathit{then}\ \mathit{if}\ u \in x \\ \qquad \mathit{then}\ u \in y. \end{array}$$

Knowing that $x < y$ or that $x \preceq_{list} y$ does not allow us to use x and y interchangeably, but it does allow us to replace certain occurrences of x with y, and vice versa.

Based on such substitutivity properties, we can introduce two deduction rules that generalize the paramodulation and E-resolution rules for equality to an arbitrary relation, under appropriate circumstances. Just as the equality rules enable us to drop the transitivity and substitutivity axioms for equality, the new relation rules enable us to drop the corresponding troublesome axioms for the relations of our theory.

POLARITY

For the equality relation, knowing that $x = y$ allows us to replace in a given sentence any occurrence of x with y and any occurrence of y with x, obtaining a sentence that follows from the given one. For an arbitrary binary relation $\prec$, knowing that $x \prec y$ still may allow us to replace certain occurrences of x with y and certain occurrences of y with x. We describe a syntactic procedure that, for a given relation $\prec$, identifies which occurrences of x and y in a given sentence can be replaced, provided we know that $x \prec y$.

More precisely, we identify particular occurrences of subexpressions of a given sentence as being positive $(+)$, negative $(-)$, or both, or neither, with respect to $\prec$. If $x \prec y$, positive occurrences of x can be replaced with y, and negative occurrences of y can be replaced with x. In other words, we can establish the substitutivity property that, for any sentence $\mathcal{P}\langle x^+,\, y^-\rangle$, the sentence

$$\begin{array}{l} \mathit{if}\ x \prec y \\ \mathit{then}\ \mathit{if}\ \mathcal{P}\langle x^+,\, y^-\rangle\ \mathit{then}\ \mathcal{P}\langle y^+,\, x^-\rangle \end{array}$$

is valid (over the theory in question). Here $\mathcal{P}\langle y^+,\, x^-\rangle$ is the sentence obtained from $\mathcal{P}\langle x^+,\, y^-\rangle$ by replacing certain positive occurrences of x with y and certain negative occurrences of y with x. With respect to the equality relation, every subexpression is both positive and negative; therefore, if we take $\prec$ to be $=$, this property reduces to the substitutivity of equality.

Our new rules are based on the above substitutivity property just as the equality rules are based on the substitutivity of equality. The new rules, like the equality rules, allow us to perform in a single application inferences that would require many steps in a conventional system. Proofs are shorter and closer to an intuitive argument; the search space is condensed accordingly.

NONCLAUSAL DEDUCTION

The paramodulation and E-resolution rules are formulated for sentences in clausal form (a disjunction of atomic sentences and their negations); on the other hand, the two corresponding rules we introduce apply to free-form sentences, with a full set of logical connectives (cf. Manna and Waldinger [80], Murray [82], Stickel [82]). By adopting such a nonclausal system, we avoid the proliferation of sentences and the disintegration of intuition that accompany the translation to clausal form. Also, it is awkward to express the mathematical induction principle in a clausal system, because we must do induction on sentences that may require more than one clause to express. On the other hand, our rules are also immediately and directly applicable to clausal theorem-proving systems.

OUTLINE

In the following section, **Preliminaries**, we sketch the basic concepts of logic that we use in this paper and we briefly outline a nonclausal deduction system. Readers who are familiar with this material should skim the section anyway, to become acquainted with our terminology and notations.

In **Relational Polarity** we introduce our central notion, the polarity of a subexpression of a sentence with respect to a given relation.

We then describe, in **The Relation Replacement Rule**, a new deduction rule that allows us to replace a subexpression of a sentence with another expression, under a wide variety of circumstances. This is our generalization of the paramodulation rule.

The rules in our system can be applied when two subexpressions can be unified. However, our second deduction rule, described in **The Relation Matching Rule**, allows us to draw a conclusion even though two subexpressions fail to unify. (Typically this rule is applied when the two subexpressions "nearly" unify.) This is our generalization of the E-resolution rule.

In **Strengthening** we tighten up our theory of polarity to allow the relation replacement rule to draw a stronger conclusion, in many circumstances.

In **Extensions**, we indicate how the notions in this paper can be extended to apply to sentences which contain explicit quantifiers and to define polarity with respect to functions as well as relations; we develop more general, conditional versions of all the rules; and we show how our results apply to problems in automated planning.

2. PRELIMINARIES

Before we can define our central notion, that of polarity of a subexpression with respect to a relation, we must introduce some concepts and notations. We will be brief and informal, because we believe that this material will be familiar to most readers.

EXPRESSIONS

We consider *terms* composed (in the usual way) of the following symbols:

- The constant symbols $a, b, c, a_1, \ldots, s, t$, and special constants such as 0.
- The variable symbols $u, v, w, x, y, u_1, \ldots$.
- The n-ary function symbols $f, g, h, f_1, \ldots$ and special symbols such as $+$.

Thus $a, x, f(a, x)$, and $f(a, x) + 0$ are terms.

We consider *propositions* composed (in the usual way) from terms and the following symbols:

- The truth symbols (logical constants) *true* and *false*.
- The n-ary relation symbols $p, q, r, p_1, \ldots$ and special symbols such as $=$ and $<$.

Thus *true* and $p(a, g(x))$ are propositions.

We consider *sentences* composed (in the usual way) from propositions and the following symbols:

- The logical connectives *not, and, or, if-then,* $\equiv$ (*if-and-only-if*), *if-then-else.*

Thus $(a < 0)$ *or* $not(p(a,\ g(x)))$ is a sentence.

The *operators* consist of the function and the relation symbols. The *expressions* consist of the terms and the sentences. Note that we do not include the quantifiers $\forall$ and $\exists$ in our language. The *ground* expressions are those that contain no variables. The expressions that occur in a given expression are its *subexpressions*. They are said to be *proper* if they are distinct from the entire expression.

REPLACEMENT

We introduce the operation of replacing subexpressions of a given expression with other expressions. We actually have two distinct notions of replacement, depending on whether or not every occurrence of the subexpression is to be replaced.

Suppose s, t, and e are expressions, where s and t are either both sentences or both terms. If we write e as $e[s]$, then $e[t]$ denotes the expression obtained by replacing every occurrence of s in $e[s]$ with t; we call this a *total replacement*. If we write e as $e\langle s\rangle$, then $e\langle t\rangle$ denotes the expression obtained by replacing certain (perhaps none) of the occurrences of s in $e\langle s\rangle$ with t; we call this a *partial replacement*.

When we say we replace certain (perhaps none) of the occurrences of s, we mean that we replace zero, one, or more occurrences. We do not require that $e[s]$ or $e\langle s\rangle$ actually contain any occurrences of s; if not, $e[t]$ and $e\langle t\rangle$ are the same as $e[s]$ and $e\langle s\rangle$, respectively. Also, while the result of a total replacement is unique, a partial replacement can produce any of several expressions.

For example, if $e[s]$ is $p(s,s,b)$, then $e[t]$ is $p(t,t,b)$. On the other hand, if $e\langle s\rangle$ is $p(s,s,b)$, then $e\langle t\rangle$ could be any of $p(s,s,b)$, $p(t,s,b)$, $p(s,t,b)$, or $p(t,t,b)$. If we want to be more specific about which occurrences are replaced, we must do so in words.

A partial replacement is *invertible*, in the sense that any sentence $e\langle s\rangle$ can be retrieved by replacing certain occurrences of t in $e\langle t\rangle$ with s. The occurrences of t to be replaced are precisely the ones introduced in obtaining $e\langle t\rangle$ in the first place. For example, if $e\langle s\rangle$ is $p(s,s,t)$, and $e\langle t\rangle$ is $p(s,t,t)$, then $e\langle s\rangle$ can be retrieved by replacing the newly introduced occurrence of t in $e\langle t\rangle$ with s.

Total replacement, on the other hand, is not invertible in the same sense. For example, if $e[s]$ is $p(s,s,t)$, then $e[t]$ is $p(t,t,t)$, and $e[s]$ cannot be obtained from $e[t]$ by replacing every occurrence of t in $e[t]$ with s.

MULTIPLE REPLACEMENT

We can extend the definition to allow the replacement of several subexpressions at once:

Suppose $s_1, \ldots, s_n, t_1, \ldots, t_n$, and e are expressions, where the s_i are distinct and, for each i, s_i and t_i are either both sentences or both terms. If we write e as $e[s_1, \ldots, s_n]$, then $e[t_1, \ldots, t_n]$ denotes the expression obtained by replacing simultaneously every occurrence of each expression s_i in e with the corresponding expression t_i; we call this a *multiple total replacement*. If we write e as $e\langle s_1, \ldots, s_n\rangle$, then $e\langle t_1, \ldots, t_n\rangle$ denotes any of the expressions obtained by replacing simultaneously certain (perhaps none) of the occurrences of some of the expressions s_i in e with the corresponding expression t_i; we call this a *multiple partial replacement*.

The replacements are made simultaneously in a single stage. For example, if $e[a,\ b]$ is $f(a,b)$, then $e[b,\ c]$ is $f(b,c)$. On the other hand, if $e\langle a,\ b\rangle$ is $f(a,b)$, then $e\langle b,\ c\rangle$ could denote any of $f(a,b)$, $f(b,b)$, $f(a,c)$, or $f(b,c)$. Even though a is replaced by b and b is replaced by c, the newly introduced occurrences of b are not replaced by c.

The replacements are made from the top down. For example, if $e[p(a,b),\ a]$ is $p(a,b)$, then $e[true,\ b]$ is *true*. We replace both $p(a,b)$ and a, but a is a subexpression of $p(a,b)$. In such cases, by convention, it is the outermost subexpression that is replaced. (For the corresponding partial replacement, either subexpression can be replaced.)

By attaching a numerical superscript, we can specify exactly how many subexpression occurrences are to be replaced in a partial replacement. Suppose $s_1, \ldots, s_n, t_1, \ldots, t_n$, and $e\langle s_1, \ldots, s_n\rangle$ are expressions and k is a nonnegative integer, where the s_i are distinct and, for each i, s_i and t_i are either both sentences or both terms. Then $e\langle t_1, \ldots, t_n\rangle^k$ is the result of replacing in $e\langle s_1, \ldots, s_n\rangle$ precisely k occurrences of $s_1, \ldots, s_n$ with the corresponding expression $t_1, \ldots, t_n$. [We assume that at least k occurrenes exist.]

Note that precisely k occurrences are replaced altogether. For example, suppose $e\langle a, b\rangle$ is $c \leq f(a, a, b)$; then $e\langle a+1, b+1\rangle^2$ could denote any of

$$c \leq f(a+1, a+1, b), \quad c \leq f(a+1, a, b+1), \quad \text{or} \quad c \leq f(a, a+1, b+1),$$

but not

$$c \leq f(a+1, a+1, b+1) \quad \text{or} \quad c \leq f(a+1, a, b).$$

We may also write $e\langle t_1, t_2, \ldots, t_n\rangle^{k,\ell}$ to indicate that precisely k or ℓ replacements are made in the expression $e\langle s_1, s_2, \ldots, s_n\rangle$.

SUBSTITUTIONS

We have a special notation for a substitution, indicating the total replacement of variables with terms. A theory of substitutions was developed by Robinson [65], in the paper in which the resolution principle was introduced. A fuller exposition of this theory appears in Manna and Waldinger [81].

For any distinct variables $x_1, x_2, \ldots, x_n$ and any terms $t_1, t_2, \ldots, t_n$, a *substitution*

$$\theta: \ \{x_1 \leftarrow t_1,\ x_2 \leftarrow t_2,\ \ldots,\ x_n \leftarrow t_n\}$$

is a set of replacement pairs $x_i \leftarrow t_i$. Note that there are no substitutions of form $\{x \leftarrow a, x \leftarrow b, \ldots\}$, where a and b are distinct. (If a and b are identical, then the set $\{x \leftarrow a, x \leftarrow a, \ldots\}$ is the same as the set $\{x \leftarrow a, \ldots\}$.) The *empty substitution* $\{\,\}$ is the set of no replacement pairs.

For any substitution θ and expression e, we denote by $e\theta$ the expression obtained by *applying* θ *to* e, i.e., by simultaneously replacing every occurrence of the variable x_i in e with the expression t_i, for each replacement pair $x_i \leftarrow t_i$ in θ. We also say that $e\theta$ is an *instance* of e. For example,

$$p(x, y)\{x \leftarrow y,\ y \leftarrow a\} \ = \ p(y, a).$$

The empty substitution $\{\,\}$ has the property that $e\{\,\} = e$ for any expression e.

Two substitutions θ and λ are said to be *equal* if they have the same effect on any expression, i.e., if, for any expression e,

$$e\theta = e\lambda.$$

For example,

$$\{x \leftarrow a,\ y \leftarrow b\} \ = \ \{x \leftarrow a,\ y \leftarrow b,\ z \leftarrow z\}.$$

Two substitutions θ and λ are equal if they agree on all variables, i.e., if $x\theta = x\lambda$ for all variables x.

For any variable x, term t, and substitution θ, the result

$$(x \leftarrow t) \circ \theta$$

of *adding* the replacement pair $x \leftarrow t$ to θ is defined to be the substitution that replaces x with t but agrees with θ on all other variables. It is thus defined by the properties

$$x\big((x \leftarrow t) \circ \theta\big) = t$$

$$y\big((x \leftarrow t) \circ \theta\big) = y\theta, \quad \text{for all variables } y \text{ distinct from } x.$$

Note that θ may already replace x with some term t'; if so, that replacement is superseded by the new one.

For example,

$$\begin{aligned}
&(y \leftarrow b) \circ \{\,\} \;=\; \{y \leftarrow b\}\\
&(x \leftarrow a) \circ \{y \leftarrow b\} \;=\; \{x \leftarrow a,\; y \leftarrow b\}\\
&(y \leftarrow c) \circ \{x \leftarrow a,\; y \leftarrow b\} \;=\; \{x \leftarrow a,\; y \leftarrow c\}\\
&(x \leftarrow x) \circ \{x \leftarrow a\} = \{\,\}.
\end{aligned}$$

We write $(x \leftarrow t) \circ (y \leftarrow t') \circ \theta$ as an abbreviation for $(x \leftarrow t) \circ \big((y \leftarrow t') \circ \theta\big)$.

The *composition* $\theta\lambda$ of two substitutions θ and λ is defined by the properties

$$\begin{aligned}
\{\,\}\lambda \;&=\; \lambda\\
\big((x \leftarrow t) \circ \theta\big)\lambda \;&=\; (x \leftarrow t\lambda) \circ (\theta\lambda)
\end{aligned}$$

for all variables x and terms t. The most important property of the composition function is that applying the composition of two substitutions θ and λ to an expression e is the same as applying first one and then the other; that is, $e(\theta\lambda) = (e\theta)\lambda$. The empty substitution can be shown to be an identity under composition; that is, $\{\,\}\theta \;=\; \theta\{\,\} \;=\; \theta$, for all substitutions θ. Also, composition can be shown to be associative; that is, $\theta(\lambda\rho) \;=\; (\theta\lambda)\rho$ for all substitutions θ, λ, and ρ.

The definition of composition suggests a way of computing it. For example,

$$\begin{aligned}
\{y \leftarrow g(z)\}\{y \leftarrow x,\; z \leftarrow b\} \;&=\; \big(y \leftarrow g(b)\big) \circ \{y \leftarrow x,\; z \leftarrow b\}\\
&=\; \{y \leftarrow g(b),\; z \leftarrow b\}
\end{aligned}$$

and therefore

$$\begin{aligned}
\{x \leftarrow y,\; y \leftarrow g(z)\}\{y \leftarrow x,\; z \leftarrow b\} \;&=\; (x \leftarrow x) \circ \{y \leftarrow g(b),\; z \leftarrow b\}\\
&=\; \{y \leftarrow g(b),\; z \leftarrow b\}.
\end{aligned}$$

Note that the composition of substitutions is not commutative. For example, $\{x \leftarrow y\}\{y \leftarrow x\} = \{y \leftarrow x\}$ and $\{y \leftarrow x\}\{x \leftarrow y\} = \{x \leftarrow y\}$, but $\{y \leftarrow x\} \neq \{x \leftarrow y\}$.

A substitution θ is said to be *more general* than a substitution θ' if there exists a substitution λ such that $\theta\lambda = \theta'$. For example, the substitution $\theta : \{x \leftarrow y\}$ is more general than the substitution $\theta' : \{x \leftarrow a, y \leftarrow a\}$, because

$$\theta\{y \leftarrow a\} \;=\; \{x \leftarrow y\}\{y \leftarrow a\} \;=\; \{x \leftarrow a, y \leftarrow a\} \;=\; \theta'.$$

On the other hand, $\theta : \{x \leftarrow y\}$ is not more general than the substitution $\phi : \{x \leftarrow a\}$, because there is no substitution λ such that

$$\theta\lambda \;=\; \{x \leftarrow y\}\lambda \;=\; \{x \leftarrow a\} \;=\; \phi.$$

A substitution is regarded as more general than itself, because $\theta\{\,\} = \theta$ for any substitution θ. It is possible for two distinct substitutions to be more general than each other. For example, $\theta : \{x \leftarrow y\}$ and $\theta' : \{y \leftarrow x\}$ are more general than each other, because

$$\theta\{y \leftarrow x\} \;=\; \{x \leftarrow y\}\{y \leftarrow x\} \;=\; \{y \leftarrow x\} \;=\; \theta'$$

and

$$\theta'\{x \leftarrow y\} \;=\; \{y \leftarrow x\}\{x \leftarrow y\} \;=\; \{x \leftarrow y\} \;=\; \theta.$$

UNIFIERS

A substitution θ is said to be a *unifier* of two expressions e and $\tilde{e}$ if

$$e\theta = \tilde{e}\theta,$$

that is, if $e\theta$ and $\tilde{e}\theta$ are identical expressions. Two expressions are *unifiable* if they have a unifier.

For example, the substitution

$$\theta :\ \{x \leftarrow b,\ y \leftarrow z\}$$

is a unifier of the two expressions

$$e : f(x, z) \qquad \mathit{and} \qquad \tilde{e} : f(b, y),$$

because $e\theta \;=\; \tilde{e}\theta \;=\; f(b, z)$. Thus, e and $\tilde{e}$ are unifiable. The substitutions

$$\phi :\ \{x \leftarrow b,\ z \leftarrow y\}$$

and

$$\rho :\ \{x \leftarrow b,\ y \leftarrow w,\ z \leftarrow w\}$$

are also unifiers of these two expressions. Thus, unifiers of expressions are not *unique*.

The expressions $p(a)$ and $p(b)$ are clearly not unifiable and neither are the expressions $q\big(x, f(x)\big)$ and $q\big(g(y), y\big)$. The expressions x and $f(x)$ are also not unifiable. Because x is a proper subexpression of $f(x)$, we know $x\theta$ is a proper subexpression of $\big(f(x)\big)\theta$, for any substitution θ; hence $x\theta$ and $\big(f(x)\big)\theta$ are not identical.

A substitution θ is said to be a *most-general unifier* of two expressions e and $\tilde{e}$ if θ is a unifier of e and $\tilde{e}$ and if θ is more general than any unifier of e and $\tilde{e}$. For example, the distinct substitutions $\theta : \{x \leftarrow b, y \leftarrow z\}$ and $\phi : \{x \leftarrow b, z \leftarrow y\}$ are both most general unifiers of the expressions $e : f(x, z)$ and $\tilde{e} : f(b, y)$. Thus, most-general unifiers are not unique. It is clear, however, that all most-general unifiers of two expressions are *equally general*, i.e., each is more general than any of the others.

There is a *unification algorithm* (Robinson [65]) for determining whether a given pair of expressions is unifiable and, if so, for producing a most general unifier.

We can extend the notion of unifier to apply to a list of pairs of expressions. A substitution θ is said to be a *simultaneous unifier* of the list

$$\langle\langle e_1, \tilde{e_1}\rangle,\ \langle e_2, \tilde{e_2}\rangle,\ \ldots,\ \langle e_n, \tilde{e_n}\rangle\rangle$$

of pairs of expressions if

$$e_1\theta = \tilde{e_1}\theta,\ \ e_2\theta = \tilde{e_2}\theta,\ \ldots,\ \ \text{and}\ \ e_n\theta = \tilde{e_n}\theta.$$

(Note that we do not require that $e_i\theta \;=\; e_j\theta$, for distinct i and j.) We may also say that θ is a *simultaneous unifier* of e_1 and $\tilde{e_1}$, of e_2 and $\tilde{e_2}$, ..., and of e_n and $\tilde{e_n}$. A list of pairs of expressions is *simultaneously unifiable* if it has a simultaneous unifier.

A list may fail to be simultaneously unifiable even though the expressions of each pair it contains are unifiable independently. For example, the list of pairs

$$\langle\langle x, g(y)\rangle,\ \langle f(x), y\rangle\rangle$$

is not simultaneously unifiable, even though the expressions x and $g(y)$ are unifiable, by the substitution $\{x \leftarrow g(y)\}$, and the expressions $f(x)$ and y are unifiable, by the substitution $\{y \leftarrow f(x)\}$.

For any list of pairs of expressions, a simultaneous unifier is *most general* if it is more general than any other simultaneous unifier.

We can extend the notion of unifier further to apply to a list of lists of expressions. A substitution θ is said to be a *simultaneous unifier* of the list

$$\langle\langle e_1, \tilde{e}_1, \tilde{\tilde{e}}_1, \ldots\rangle, \ \langle e_2, \tilde{e}_2, \tilde{\tilde{e}}_2, \ldots\rangle, \ \ldots, \ \langle e_n, \tilde{e}_n, \tilde{\tilde{e}}_n, \ldots\rangle\rangle,$$

of lists of expressions if

$$\begin{aligned} e_1\theta &= \tilde{e}_1\theta = \tilde{\tilde{e}}_1\theta = \ldots \\ e_2\theta &= \tilde{e}_2\theta = \tilde{\tilde{e}}_2\theta = \ldots \\ &\vdots \\ e_n\theta &= \tilde{e}_n\theta = \tilde{\tilde{e}}_n\theta = \ldots. \end{aligned}$$

We may also say that θ is a simultaneous unifer of $e_1, \tilde{e}_1, \tilde{\tilde{e}}_1, \ldots$, of $e_2, \tilde{e}_2, \tilde{\tilde{e}}_2, \ldots$, and of $e_n, \tilde{e}_n, \tilde{\tilde{e}}_n, \ldots$. The notion of most-general simultaneous unifier and the unification algorithm may be extended accordingly. The notation is more complex but the concepts are the same.

SUBSTITUTION AND REPLACEMENT

We sometimes find it convenient to use the replacement and substitution notations together. Suppose s, t, and e are expressions, where s and t are either both sentences or both terms. Let θ be a substitution. If we write e as $e[s]$, then

$$e\theta[t]$$

denotes the expression obtained by replacing every occurrence of $s\theta$ in $e\theta$ with t. If we write e as $e\langle s\rangle$, then

$$e\theta\langle t\rangle$$

denotes the expression obtained by replacing certain (perhaps none) of the occurrences of $s\theta$ in $e\theta$ with t.

For example, consider the expression

$$e: \quad p\big(f(x,a)\big) \ \mathit{or} \ q\big(f(x,y)\big) \ \mathit{or} \ r\big(f(b,a)\big)$$

and the substitution

$$\theta: \ \{x \leftarrow b, \ y \leftarrow a\}.$$

If we write e as $e[f(x,a)]$, then $e\theta[g(c)]$ is

$$p\big(g(c)\big) \ \mathit{or} \ q\big(g(c)\big) \ \mathit{or} \ r\big(g(c)\big).$$

Note that two of the replaced occurrences of $f(x,a)\theta$ in $e\theta$ do not correspond to occurrences of $f(x,a)$ in e; they were created by application of the substitution θ.

INTERPRETATIONS

We shall use the Herbrand notion of interpretation, in which the elements of the domain are identified with the terms of the language.

An *interpretation* $\mathcal{I}$ is an assignment of truth values, either T (true) or F (false), to every ground proposition (i.e., to every proposition that contains no variables). If $\mathcal{I}$ assigns T [or F] to a ground proposition, that proposition is said to be *true* [or *false*] *under* $\mathcal{I}$. The truth [or falseness] of a nonpropositional ground sentence under an interpretation $\mathcal{I}$ may be determined from that of its propositional constituents by the familiar semantic rules for the logical connectives.

A nonground sentence $\mathcal{P}$ is *true under* $\mathcal{I}$ if every ground instance of $\mathcal{P}$ is true under $\mathcal{I}$; otherwise, $\mathcal{P}$ is *false under* $\mathcal{I}$. Note that, according to this definition, free variables have a tacit universal quantification.

We can now define the notions of implication and equivalence between sentences. The sentences $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \ldots$ *imply* a sentence $\mathcal{Q}$ if, for any interpretation $\mathcal{I}$,

> if $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_2, \ldots$ are all true under $\mathcal{I}$,
> then $\mathcal{Q}$ is true under $\mathcal{I}$.

Note that if $\mathcal{P}$ implies $\mathcal{Q}$, it is not necessarily the case that the sentence (*if* $\mathcal{P}$ *then* $\mathcal{Q}$) is valid. For example, $p(x)$ implies $p(a)$, because free variables are taken to be universally quantified. But the sentence (*if* $p(x)$ *then* $p(a)$) is not valid: its instance (*if* $p(b)$ *then* $p(a)$) is false under any interpretation for which $p(b)$ is true and $p(a)$ is false.

Two sentences $\mathcal{P}$ and $\mathcal{Q}$ are *equivalent* if, for any interpretation $\mathcal{I}$,

> $\mathcal{P}$ is true under $\mathcal{I}$
>   if and only if
> $\mathcal{Q}$ is true under $\mathcal{I}$.

Hence $\mathcal{P}$ is equivalent to $\mathcal{Q}$ if $\mathcal{P}$ implies $\mathcal{Q}$ and $\mathcal{Q}$ implies $\mathcal{P}$. For example, the sentences $p(x)$ and $p(y)$ are equivalent.

Lemma (instantiation)

For any sentence $\mathcal{F}$ and substitution θ, $\mathcal{F}$ implies $\mathcal{F}\theta$. ⌟

Both total and partial replacement exhibit the following *value* property:

Suppose $\mathcal{P}$, $\mathcal{Q}$, and $\mathcal{F}$ are ground sentences and $\mathcal{I}$ is an interpretation. Then

> if $\mathcal{P}$ and $\mathcal{Q}$ have the same truth value under $\mathcal{I}$,
> then $\mathcal{F}[\mathcal{P}]$ and $\mathcal{F}[\mathcal{Q}]$ have the same truth value under $\mathcal{I}$.

Also,

> if $\mathcal{P}$ and $\mathcal{Q}$ have the same truth value under $\mathcal{I}$,
> then $\mathcal{F}\langle\mathcal{P}\rangle$ and $\mathcal{F}\langle\mathcal{Q}\rangle$ have the same truth value under $\mathcal{I}$.

A corresponding *value* property applies to multiple replacements.

Remark

The value property applies only to ground sentences, not to sentences with variables. For instance, let $\mathcal{P}$ be the sentence $p(x)$, let $\mathcal{Q}$ be the sentence *false*, and let $\mathcal{F}[\mathcal{P}]$ be the sentence (*not* $p(x)$). Consider an interpretation $\mathcal{I}$ under which

> $p(a)$ is true and $p(b)$ is false.

Then (by the definition of truth for a nonground sentence) $p(x)$ is false under $\mathcal{I}$ and hence

> $p(x)$ and *false* have the same truth value under $\mathcal{I}$.

On the other hand (by the definition again) *not* $p(x)$ is also false under $\mathcal{I}$ and hence

$$(\textit{not}\; p(x)) \text{ and } (\textit{not false}) \text{ do not have the same truth value under } \mathcal{I},$$

contradicting the conclusion of the value property. $\lrcorner$

THEORIES

A *theory* is a set of sentences $\mathcal{T}$ that is closed under logical implication: If $\mathcal{T}$ implies a sentence $\mathcal{P}$ then $\mathcal{P}$ belongs to $\mathcal{T}$. A member of a theory $\mathcal{T}$ is also said to be *valid* in $\mathcal{T}$.

A theory $\mathcal{T}$ is said to be *defined by* a set of sentences $\mathcal{A}$ if $\mathcal{T}$ is precisely the set of sentences implied by $\mathcal{A}$. We shall also say that $\mathcal{A}$ is a set of *axioms* for $\mathcal{T}$.

An interpretation $\mathcal{I}$ is said to be a *model* for a theory $\mathcal{T}$ if every sentence of $\mathcal{T}$ is true under $\mathcal{I}$.

For example, let $\mathcal{T}$ be the set of sentences implied by the transitivity axiom,

$$\begin{array}{l} \textit{if}\ x \prec y\ \textit{and}\ y \prec z \\ \textit{then}\ x \prec z, \end{array}$$

and the *irreflexivity* axiom,

$$\textit{not}\, x \prec x.$$

Then $\mathcal{T}$ is a theory, defined by these axioms. The *asymmetry* property

$$\begin{array}{l} \textit{if}\ x \prec y \\ \textit{then}\ \textit{not}\, y \prec x \end{array}$$

is a (valid) sentence of this theory.

RELATIONS

We need some special terminology for speaking about relations. Henceforth, let us consider a particular theory. When we speak of validity, we shall mean validity in that theory.

Let p and q be n-ary relations. Then we say that p *implies* q if

$$\textit{if}\ p(x_1, x_2, \ldots, x_n)\ \textit{then}\ q(x_1, x_2, \ldots, x_n)$$

is valid (in the theory under discussion). We also say that p is *equivalent* to q if

$$p(x_1, x_2, \ldots, x_n) \equiv q(x_1, x_2, \ldots, x_n)$$

is valid.

Let $\prec$ be an arbitrary binary relation. We shall say that, over a given theory, $\prec$ is *reflexive* if

$$x \prec x$$

is valid (in the theory); $\prec$ is *irreflexive* if

$$\textit{not}\,(x \prec x)$$

is valid; $\prec$ is *total* if

$$x \prec y\ \textit{or}\ x = y\ \textit{or}\ y \prec x$$

is valid; $\mathrel{-\!\!\!\prec}$ is *transitive* if

$$if\ (x \mathrel{-\!\!\!\prec} y\ \ and\ \ y \mathrel{-\!\!\!\prec} z)\ \ then\ \ x \mathrel{-\!\!\!\prec} z$$

is valid; and $\mathrel{-\!\!\!\prec}$ is symmetric if

$$if\ \ x \mathrel{-\!\!\!\prec} y\ \ then\ \ y \mathrel{-\!\!\!\prec} x$$

is valid.

We regard logical connectives as relations on the set of truth values $\{\text{T}, \text{F}\}$. For instance, the implication connective $(if\, \mathcal{P}\ then\ \mathcal{Q})$ is the relation that holds if $\mathcal{P}$ has value F or if $\mathcal{P}$ and $\mathcal{Q}$ both have value T; we may read it as "$\mathcal{P}$ is falser than (or as false as) $\mathcal{Q}$." The equivalence connective $\mathcal{P} \equiv \mathcal{Q}$ is simply the equality relation on $\{\text{T}, \text{F}\}$. Note that, viewed as binary relations, the implication connective *if-then* is reflexive, total, and transitive, and the equivalence connective $\equiv$ is reflexive, transitive, and symmetric.

ASSOCIATED RELATIONS

For each binary relation, we shall be concerned with certain associated relations.

Consider an arbitrary binary relation $x \mathrel{-\!\!\!\prec} y$ (read as "x is related to y"). The *reflexive closure* $\mathrel{\underline{-\!\!\!\prec}}$ of $\mathrel{-\!\!\!\prec}$ is defined by

$$x \mathrel{\underline{-\!\!\!\prec}} y \quad \equiv \quad (x \mathrel{-\!\!\!\prec} y\ \ or\ \ x = y).$$

The *irreflexive* restriction $\prec$ of $\mathrel{-\!\!\!\prec}$ is defined by

$$x \prec y \quad \equiv \quad \big(x \mathrel{-\!\!\!\prec} y\ \ and\ \ not\,(x = y)\big).$$

The *inverse* $\mathrel{\succ\!\!\!-}$ of $\mathrel{-\!\!\!\prec}$ is defined by

$$x \mathrel{\succ\!\!\!-} y \quad \equiv \quad y \mathrel{\succ\!\!\!-} x.$$

The negation $\not\mathrel{-\!\!\!\prec}$ of $\mathrel{-\!\!\!\prec}$ is defined by

$$x \not\mathrel{-\!\!\!\prec} y \quad \equiv \quad not\,(x \mathrel{-\!\!\!\prec} y).$$

We use $\succ$ and $\succeq$ to denote the inverses of $\prec$ and $\preceq$, respectively, and $\not\prec$ and $\not\preceq$ to denote their negations. If we are using the prefix notation $p(x, y)$ for a binary relation, we denote its reflexive closure by $\overline{\overline{p}}(x, y)$, its irreflexive restriction by $\overset{\neq}{p}(x, y)$, and its negation by $\not p(x, y)$.

The following proposition connects the relations associated with a given binary relation:

Proposition (negation of associated relations)

Consider an arbitrary binary relation $\mathrel{-\!\!\!\prec}$.

The negation $\not\mathrel{\underline{-\!\!\!\prec}}$ of the reflexive closure of $\mathrel{-\!\!\!\prec}$ is equivalent to the irreflexive restriction of its negation $\not\mathrel{-\!\!\!\prec}$, that is,

$$x \not\mathrel{\underline{-\!\!\!\prec}} y \qquad \text{if and only if} \qquad \big(x \not\mathrel{-\!\!\!\prec} y\ \ and\ \ not\,(x = y)\big).$$

The negation $\not\prec$ of the irreflexive restriction of $\mathrel{-\!\!\!\prec}$ is equivalent to the reflexive closure of its negation $\not\mathrel{-\!\!\!\prec}$, that is,

$$x \not\prec y \qquad \text{if and only if} \qquad (x \not\mathrel{-\!\!\!\prec} y\ \ or\ \ x = y).$$

┘

3. RELATIONAL POLARITY

We are now ready to define our key notion, the polarity of a subexpression with respect to a given binary relation. We actually define the polarity of a subexpression with respect to two binary relations, $\prec_1$ and $\prec_2$. This notion is to be defined so that, if the subexpression is positive, replacing that subexpression with a larger expression (with respect to $\prec_1$) will make the entire expression larger (with respect to $\prec_2$). Similarly, if the subexpression is negative, replacing that subexpression with a smaller expression (with respect to $\prec_1$) will make the entire expression larger (with respect to $\prec_2$).

We begin by defining polarity for the arguments of an operator (i.e., function or relation).

Definition (polarity of an operator)

Let f be an n-ary operator and $\prec_1$ and $\prec_2$ be binary relations. Then

- f is *positive* over its ith argument with respect to $\prec_1$ and $\prec_2$ if the sentence

  if $x \prec_1 y$
  then $f(z_1, \ldots, z_{i-1}, x, z_{i+1}, \ldots, z_n) \prec_2 f(z_1, \ldots, z_{i-1}, y, z_{i+1}, \ldots, z_n)$

  is valid. In other words, replacing x with a larger element y makes

  $f(z_1, \ldots, z_{i-1}, x, z_{i+1}, \ldots, z_n)$

  larger.

- f is *negative* over its ith argument with respect to $\prec_1$ and $\prec_2$ if the sentence

  if $x \prec_1 y$
  then $f(z_1, \ldots, z_{i-1}, y, z_{i+1}, \ldots, z_n) \prec_2 f(z_1, \ldots, z_{i-1}, x, z_{i+1}, \ldots, z_n)$

  is valid. In other words, replacing y with a smaller element x makes

  $f(z_1, \ldots, z_{i-1}, y, z_{i+1}, \ldots, z_n)$

  larger. $\lrcorner$

We illustrate this notion with two examples.

Example

Suppose our theory includes the finite sets and the nonnegative integers. Take $f(z)$ to be the cardinality function $card(z)$, which maps each set into the number of elements it contains. Take $\prec_1$ to be the subset relation $\subseteq$ over the finite sets and $\prec_2$ to be the weak less-than relation $\leq$ over the nonnegative integers.

Then the *card* function is positive over its first (and only) argument with respect to the relations $\subseteq$ and $\leq$, because the sentence

if $x \subseteq y$
then $card(x) \leq card(y)$

is valid (in the theory). $\lrcorner$

Example

Consider the theory of the integers. Take $f(z_1, z_2)$ to be the less-than relation $z_1 < z_2$. Take $x \prec_1 y$ to be the predecessor relation $x \prec_{pred} y$, which holds if $x = y - 1$, and take $\prec_2$ to be the *if-then* connective. (Recall that we regard connectives as relations on the set of truth values.)

Then the less-than relation $<$ is negative over its first argument with respect to $\prec_{pred}$ and *if-then*, because the sentence

$$\begin{array}{l} \textit{if } x \prec_{pred} y \\ \textit{then if } y < z_2 \ \textit{ then } x < z_2 \end{array}$$

is valid. Also, $<$ is positive over its second argument with respect to $\prec_{pred}$ and *if-then*, because the sentence

$$\begin{array}{l} \textit{if } x \prec_{pred} y \\ \textit{then if } z_1 < x \ \textit{ then } z_1 < y \end{array}$$

is valid. $\lrcorner$

It follows from the definition that, for any n-ary operator f and binary relations $\prec_1$ and $\prec_2$,

f is positive over its ith argument with respect to $\prec_1$ and $\prec_2$
if and only if
f is negative over its ith argument with respect to $\succ_1$ and $\prec_2$
if and only if
f is negative over its ith argument with respect to $\prec_1$ and $\succ_2$
if and only if
f is positive over its ith argument with respect to $\succ_1$ and $\succ_2$.

When we say that a relation $p(z_1, \ldots, z_n)$ is positive or negative over its ith argument with respect to a single relation $\prec_1$, without mentioning a second relation $\prec_2$, we shall by convention take $\prec_2$ to be the *if-then* connective. Thus in the above example we may simply say that $<$ is negative over its first argument and positive over its second argument, with respect to $\prec_{pred}$.

Every relation is both positive and negative over each of its arguments with respect to the equality relation $=$, because the sentences

$$\begin{array}{l} \textit{if } x = y \\ \textit{then if } p(z_1, \ldots, x, \ldots, z_n) \\ \qquad \textit{then } p(z_1, \ldots, y, \ldots, z_n) \end{array} \quad \text{and} \quad \begin{array}{l} \textit{if } x = y \\ \textit{then if } p(z_1, \ldots, y, \ldots, z_n) \\ \qquad \textit{then } p(z_1, \ldots, x, \ldots, z_n) \end{array}$$

are valid. This is equivalent to the relational-substitutivity property of equality. Also, every function is both positive and negative over each of its arguments with respect to $=$ and $=$, because the sentences

$$\begin{array}{l} \textit{if } x = y \\ \textit{then } f(z_1, \ldots, x, \ldots, z_n) = f(z_1, \ldots, y, \ldots, z_n) \end{array} \quad \text{and} \quad \begin{array}{l} \textit{if } x = y \\ \textit{then } f(z_1, \ldots, y, \ldots, z_n) = f(z_1, \ldots, x, \ldots, z_n) \end{array}$$

are valid. This is equivalent to the functional-substitutivity property of equality.

Every connective is both positive and negative over all its arguments with respect to $\equiv$. For example, the *not* connective is both positive and negative over its argument with respect to $\equiv$, because both sentences

$$\begin{array}{l} \textit{if } x \equiv y \\ \textit{then if } (\textit{not } x) \textit{ then } (\textit{not } y) \end{array} \quad \text{and} \quad \begin{array}{l} \textit{if } x \equiv y \\ \textit{then if } (\textit{not } y) \textit{ then } (\textit{not } x) \end{array}$$

are valid.

When we say that a connective is positive or negative over its ith argument, without mentioning any relations $\prec_1$ and $\prec_2$ at all, we shall by convention take both $\prec_1$ and $\prec_2$ to be the *if-then* connective. Polarity in this sense is close to its ordinary use in logic. The negation connective *not* is negative in its first (and only) argument, because the sentence

$$\begin{array}{l} \textit{if if } x \textit{ then } y \\ \textit{then if } (\textit{not } y) \textit{ then } (\textit{not } x) \end{array}$$

is valid. The conjunction connective *and* and the disjunction connective *or* are positive over both their arguments. The implication connective *if-then* is negative in its first argument, but positive in its second. The equivalence connective $\equiv$ has no polarity in either argument. The conditional connective *if-then-else* has no polarity in its first argument, but is positive in its second and third argument.

Note that a binary relation $\prec$ is transitive if and only if it is negative with respect to $\prec$ itself over its first argument, because the polarity condition

$$\begin{array}{l} \textit{if } x \prec y \\ \textit{then if } y \prec z \textit{ then } x \prec z \end{array}$$

is equivalent to the definition of transitivity. Also, $\prec$ is transitive if and only if it is positive with respect to $\prec$ over its second argument.

We are now ready to define polarity for the subexpressions of a given expression. The definition is inductive.

Definition (polarity of a subexpression)

Let $\prec_1$ and $\prec_2$ be binary relations. Then

- An expression s *is positive in* s itself with respect to $\prec_1$ and $\prec_2$ if $\prec_1$ implies $\prec_2$.
- An expression s *is negative in* s itself with respect to $\prec_1$ and $\prec_2$ if $\prec_1$ implies $\succ_2$.

Let f be an n-ary operator and $e_1, e_2, \ldots, e_n$ be expressions. Consider an occurrence of s in one of the expressions e_i. Then

- The occurrence of s is *positive* in $f(e_1, e_2, \ldots, e_n)$ with respect to $\prec_1$ and $\prec_2$ if there exists a binary relation $\prec$ such that

  the polarity of the occurrence of s in e_i with respect to $\prec_1$ and $\prec$
  is the same as
  the polarity of f over its ith argument with respect to $\prec$ and $\prec_2$.

- The occurrence of s is *negative* in $f(e_1, e_2, \ldots, e_n)$ with respect to $\prec_1$ and $\prec_2$ if there exists a binary relation $\prec$ such that

  the polarity of the occurrence of s in e_i with respect to $\prec_1$ and $\prec$
  is opposite to
  the polarity of f over its ith argument with respect to $\prec$ and $\prec_2$.

Furthermore, if f has no polarity over its ith argument or if s has no polarity in e_i, then s has no polarity in $f(e_1, e_2, \ldots, e_n)$. On the other hand, if s has both polarities in e_i and f has some polarity over its i argument, or if f has both polarities over its ith argument and s has some polarity in e_i, then s automatically has both polarities in $f(e_1, e_2, \ldots, e_n)$. ⌟

Remark

For any binary relation $\prec$, any expression s is positive in s itself with respect to $\prec$ and $\prec$ (because $\prec$ implies $\prec$). Similarly, s is negative in s with respect to $\prec$ and $\succ$.

If f is positive over its ith argument with respect to $\prec_1$ and $\prec_2$, then, for any expressions $e_1, e_2, \ldots, e_n$, the occurrence of e_i is positive in $f(e_1, \ldots, e_i, \ldots, e_n)$ with respect to $\prec_1$ and $\prec_2$. For take $\prec$ to be $\prec_1$. Then the polarity of e_i in e_i itself is positive with respect to $\prec_1$ and $\prec_1$. Also, f is positive over its ith argument with respect to $\prec_1$ and $\prec_2$. Because these two polarities are the same, e_i is positive in $f(e_1, \ldots, e_i, \ldots, e_n)$ with respect to $\prec_1$ and $\prec_2$.

Similarly, if f is negative over its ith argument, then e_i is negative in $f(e_1, \ldots, e_i, \ldots, e_n)$, with respect to $\prec_1$ and $\prec_2$. $\lrcorner$

We may indicate the polarity of a subexpression s by annotating it s^+, s^-, or $s^\pm$.

For example, suppose our theory includes the theories of sets and nonnegative integers. The occurrence of s in the sentence

$$card(s^-) < m$$

is negative with respect to the subset relation $\subseteq$ and the *if-then* connective. For note that *card* is positive over its argument with respect to $\subseteq$ and $\leq$ and that $<$ is negative over its first argument with respect to $\leq$ and *if-then*. Therefore, by our remark, we know that s is positive in $card(s)$ with respect to $\subseteq$ and $\leq$ and that $card(s)$ is negative in $card(s) < m$ with respect to $\leq$ and *if-then*. By the definition, taking $\prec_1$ to be $\subseteq$, $\prec$ to be $\leq$, and $\prec_2$ to be *if-then*, we conclude that s is negative in $card(s) < m$ with respect to $\subseteq$ and *if-then*.

When we say that an occurrence of a subexpression is positive or negative in a sentence with respect to a single relation $\prec_1$, without mentioning a second relation $\prec_2$, we shall again take $\prec_2$ to be the *if-then* connective. When we say that an occurrence of a subsentence is positive or negative in a sentence, without mentioning any relation at all, we shall again take both $\prec_1$ and $\prec_2$ to be *if-then*.

It can be established from the definition that, for expressions s and t and binary relations $\prec_1$ and $\prec_2$,

> an occurrence of s is positive in t with respect to $\prec_1$ and $\prec_2$

if and only if

> the occurrence of s is negative in t with respect to $\succ_1$ and $\prec_2$

if and only if

> the occurrence of s is negative in t with respect to $\prec_1$ and $\succ_2$

if and only if

> the occurrence of s is positive in t with respect to $\succ_1$ and $\succ_2$.

This is analogous to our previous result concerning polarity for the argument of an operator.

Suppose an occurrence of s is positive [or negative] in t with respect to $\prec_1$ and $\prec_2$. Then if $\tilde{\prec}_1$ is a binary relation that implies $\prec_1$, then s is positive [or negative, respectively] in t with respect to $\tilde{\prec}_1$ and $\prec_2$. Similarly, if $\prec_2$ implies a binary relation $\tilde{\prec}_2$, then s is positive [or negative, respectively] in t with respect to $\prec_1$ and $\tilde{\prec}_2$.

We can establish the following result:

Lemma (polarity operator)

Let $\prec_1$ and $\prec_2$ be binary relations, f be an n-ary operator, and $e_1, e_2, \ldots, e_n$ be expressions. Consider an occurrence of s in one of the expressions e_i such that s has some polarity in $f(e_1, e_2, \ldots, e_n)$ with respect to $\prec_1$ and $\prec_2$.

Then there exists a binary relation $\prec$ such that

> f is positive over its ith argument with respect to $\prec$ and $\prec_2$

and

> the polarity of the occurrence of s in $f(e_1, e_2, \ldots, e_n)$ with respect to $\prec_1$ and $\prec_2$
> is the same as
> the polarity of the occurrence of s in e_i with respect to $\prec_1$ and $\prec$. $\lrcorner$

Proof

Consider the case in which the occurrence of s is positive in $f(e_1, e_2, \ldots, e_n)$ with respect to $\prec_1$ and $\prec_2$. According to the definition, this means that there exists a binary relation $\tilde{\prec}$ such that

> the polarity of the occurrence of s in e_i with respect to $\prec_1$ and $\tilde{\prec}$
> is the same as
> the polarity of f over its ith argument with respect to $\tilde{\prec}$ and $\prec_2$.

If f is positive over its ith argument with respect to $\tilde{\prec}$ and $\prec_2$, then the occurrence of s is positive in e_i with respect to $\prec_1$ and $\tilde{\prec}$, and we can simply take $\prec$ to be $\tilde{\prec}$.

On the other hand, if f is negative over its ith argument with respect to $\tilde{\prec}$ and $\prec_2$, then the occurrence of s is negative in e_i with respect to $\prec_1$ and $\tilde{\prec}$. By previous remarks, this means that f is positive over its ith argument with respect to the inverse relation $\tilde{\succ}$ and $\prec_2$, and the occurrence of s is positive in e_i with respect to $\prec_1$ and the inverse relation $\tilde{\succ}$. Hence we can take $\prec$ to be $\tilde{\succ}$.

The case in which s is negative in $f(e_1, e_2, \ldots, e_n)$ is treated similarly. ┛

Polarities of subexpressions of subexpressions can be composed according to the following result.

Lemma (polarity composition)

Consider an occurrence of a subexpression r in an expression s and an occurrence of s in an expression t. Then the polarity of the occurrence of r is positive [or negative] in t with respect to binary relations $\prec_1$ and $\prec_2$ if and only if there exists a binary relation $\prec$ such that

> the polarity of the occurrence of r in s with respect to $\prec_1$ and $\prec$
> is the same as [or opposite to, respectively]
> the polarity of the occurrence of s in t with respect to $\prec$ and $\prec_2$. ┛

For instance, if r is negative in s and s is negative in t then r is positive in t, with respect to the appropriate binary relations. If r has both polarities in s and s has some polarity in t, then r has both polarities in t.

We can now establish the fundamental property of polarity.

Lemma (polarity replacement)

For any binary relations $\prec_1$ and $\prec_2$ and expression $e\langle x^+,\ y^-\rangle$, the sentence

$$\begin{array}{l} \textit{if } x \prec_1 y \\ \textit{then } e\langle x^+,\ y^-\rangle \prec_2 e\langle y^+,\ x^-\rangle^1 \end{array}$$

is valid. Here $e\langle y^+,\ x^-\rangle^1$ is the result of replacing in $e\langle x^+,\ y^-\rangle$ precisely one positive occurrence of x with y or negative occurrence of y with x (we assume that such an occurrence exists) where the polarity is taken in $e\langle x^+,\ y^-\rangle$ with respect to $\prec_1$ and $\prec_2$. ┛

Example

Suppose our theory includes the theories of lists and nonnegative integers. Take $\prec_1$ to be the tail relation $x \prec_{tail} y$, which is true if

$$\textit{not}\,(y = [\,])\ \textit{ and }\ x = tail(y),$$

that is, if y is nonempty and x is the list of all but the first element of y. Take $\prec_2$ to be the predecessor relation $\prec_{pred}$. Take $e\langle x^+,\, y^-\rangle$ to be the expression

$$length(x^+) + length(x^+),$$

where the function $length(x)$ yields the number of elements in the list x.

Note that each occurrence of x is positive in $length(x) + length(x)$ with respect to $\prec_{tail}$ and $\prec_{pred}$, as indicated by the annotations. For, each occurrence is positive in $length(x)$ with respect to $\prec_{tail}$ and $\prec_{pred}$, and the plus function $+$ is positive over either of its arguments with respect to $\prec_{pred}$ and $\prec_{pred}$.

Therefore, according to the lemma, the sentence

if $x \prec_{tail} y$
then $length(x) + length(x) \prec_{pred} length(y) + length(x)$

is valid, because $length(y) + length(x)$ is the result of replacing one positive occurrence of x in $length(x) + length(x)$ with y. Also, according to the lemma, the sentence

if $x \prec_{tail} y$
then $length(x) + length(x) \prec_{pred} length(x) + length(y)$

is valid, because $length(x) + length(y)$ is the result of replacing one positive occurrence of x in $length(x) + length(x)$ with y.

On the other hand, the lemma does not allow us to conclude that

if $x \prec_{tail} y$
then $length(x) + length(x) \prec_{pred} length(y) + length(y)$

is valid, because $length(y) + length(y)$ is obtained by replacing two, not one, positive occurrences of x in $length(x) + length(x)$ with y. In fact, this third sentence is not valid. ⌟

We now prove the lemma.

Proof (polarity replacement lemma)

For any arbitrary binary relation $\prec_1$, suppose that

$$x \prec_1 y.$$

We show that, for any expression $e\langle x^+,\, y^-\rangle$, we have, for any binary relation $\prec_2$,

$$e\langle x^+,\, y^-\rangle \prec_2 e\langle y^+,\, x^-\rangle^1.$$

The proof is by induction on the structure of $e\langle x^+,\, y^-\rangle$. In other words, we show the desired conclusion for an arbitrary expression $e\langle x^+,\, y^-\rangle$, under the induction hypothesis that, for any proper subexpression $\tilde{e}\langle x^+,\, y^-\rangle$ of $e\langle x^+,\, y^-\rangle$, we have, for any binary relation $\tilde{\prec}_2$,

$$\tilde{e}\langle x^+,\, y^-\rangle \;\tilde{\prec}_2\; \tilde{e}\langle y^+,\, x^-\rangle^1.$$

As in the statement of the lemma, $\tilde{e}\langle y^+, x^-\rangle^1$ is obtained from $\tilde{e}\langle x^+, y^-\rangle$ by replacing precisely one occurrence of x or y, of suitable polarity with respect to $\prec_1$ and $\tilde{\prec}_2$.

The proof distinguishes among several subcases.

Case: The expression $e\langle x^+,\, y^-\rangle$ is simply x

Then, because the replaced variable x is positive in x, with respect to $\prec_1$ and $\prec_2$, we have (by the definition of polarity) that $\prec_1$ implies $\prec_2$.

In this case, $e\langle y^+,\ x^-\rangle^1$ is y, and we must show

$$x \prec_2 y.$$

But this follows from our supposition that $x \prec_1 y$, because $\prec_1$ implies $\prec_2$.

Case: The expression $e\langle x^+,\ y^-\rangle$ is simply y

Then, because the replaced variable y is negative in y with respect to $\prec_1$ and $\prec_2$, we have (by the definition of polarity) that $\prec_1$ implies $\succ_2$.

In this case, $e\langle y^+,\ x^-\rangle$ is x, and we must show that

$$y \prec_2 x,$$

or, equivalently, that

$$x \succ_2 y.$$

But this follows from our supposition that $x \prec_1 y$, because $\prec_1$ implies $\succ_2$.

Case: $e\langle x^+,\ y^-\rangle$ is of form $f(e_1, e_2, \ldots, e_n)$, where f is an n-ary operator

The replaced occurrence of x [or y] must occur in one of the arguments e_i of f. Because this occurrence is positive [or negative, respectively] in $f(e_1, e_2, \ldots, e_n)$ with respect to $\prec_1$ and $\prec_2$, we know (by the *polarity operator* lemma) that there exists a binary relation $\prec$ such that

f is positive over its ith argument with resect to $\prec$ and $\prec_2$

and

the polarity of the replaced occurrence of x [or y] in e_i with respect to $\prec_1$ and $\prec$
is the same as
the polarity of the replaced occurrence of x [or y] in $f(e_1, e_2, \ldots, e_n)$, that is, $e\langle x^+,\ y^-\rangle$, with respect to $\prec_1$ and $\prec_2$.

Let us therefore write e_i as $e_i\langle x^+,\ y^-\rangle$.

Because $e_i\langle x^+,\ y^-\rangle$ is a proper subexpression of $e\langle x^+,\ y^-\rangle$, we can apply our induction hypothesis, taking $\tilde{e}\langle x^+,\ y^-\rangle$ to be $e_i\langle x^+,\ y^-\rangle$ and $\tilde{\prec}_2$ to be $\prec$, to conclude that

$$e_i\langle x^+,\ y^-\rangle \prec e_i\langle y^+,\ x^-\rangle^1.$$

Therefore (by the definition of polarity of an operator, because f is positive over its ith argument with respect to $\prec$ and $\prec_2$), we have

$$f(e_1, \ldots, e_i\langle x^+,\ y^-\rangle, \ldots, e_n) \prec_2 f(e_1, \ldots, e_i\langle y^+,\ x^-\rangle, \ldots, e_n),$$

that is,

$$e\langle x^+,\ y^-\rangle \prec_2 e\langle y^+,\ x^-\rangle^1,$$

as we wanted to show. This completes the proof. $\lrcorner$

The polarity replacement lemma allows us to replace precisely one occurrence of a variable. If we know more about the relation $\prec_2$, we can establish stronger versions of the lemma. In particular, if we know that $\prec_2$ is transitive, we can replace one or more occurrences of the variable.

Lemma (transitive polarity replacement)

For any binary relations $\prec_1$ and $\prec_2$ and expression $e\langle x^+,\, y^-\rangle$, where $\prec_2$ is transitive, the sentence

$$\begin{aligned}&\textit{if } x \prec_1 y\\ &\textit{then } e\langle x^+,\, y^-\rangle \prec_2 e\langle y^+,\, x^-\rangle^n\end{aligned}$$

is valid for every positive integer n. Here $e\langle y^+,\, x^-\rangle^n$ is the result of replacing in $e\langle x^+,\, y^-\rangle$ precisely n positive occurrences of x with y or negative occurrences of y with x, where the polarity is taken in $e\langle x^+,\, y^-\rangle$ with respect to $\prec_1$ and $\prec_2$. ∎

Note that we can replace occurrences of both x and y in the same expression; precisely n replacements are made altogether. Also, the lemma requires that at least one replacement be made.

Example

Suppose our theory includes the theories of both lists and integers. Take $e\langle x^+,\, y^-\rangle$ to be the expression

$$e\langle x^+,\, y^-\rangle : \quad \textit{length}(x^+) + \big(\textit{length}(x^+) - \textit{length}(y^-)\big).$$

Take $\prec_1$ to be the tail relation $\prec_{tail}$ (defined in a previous example) and $\prec_2$ to be the less-than relation $<$. Note that, with respect to $\prec_{tail}$ and $<$, both occurrences of x are positive and the occurrence of y is negative in $e\langle x^+,\, y^-\rangle$; also $<$ is transitive. According to the lemma, the following sentences (among others) are valid: the sentence

$$\begin{aligned}&\textit{if } x \prec_{tail} y\\ &\textit{then } \textit{length}(x) + \big(\textit{length}(x) - \textit{length}(y)\big) < \textit{length}(y) + \big(\textit{length}(y) - \textit{length}(y)\big).\end{aligned}$$

for which both occurrences of x in $e\langle x^+,\, y^-\rangle$ have been replaced, and

$$\begin{aligned}&\textit{if } x \prec_{tail} y\\ &\textit{then } \textit{length}(x) + \big(\textit{length}(x) - \textit{length}(y)\big) < \textit{length}(x) + \big(\textit{length}(y) - \textit{length}(x)\big),\end{aligned}$$

for which one occurrence of x and one of y in $e\langle x^+,\, y^-\rangle$ have been replaced.

On the other hand, the lemma does not allow us to conclude that

$$\begin{aligned}&\textit{if } x \prec_{tail} y\\ &\textit{then } \textit{length}(x) + \big(\textit{length}(x) - \textit{length}(y)\big) < \textit{length}(x) + \big(\textit{length}(x) - \textit{length}(y)\big),\end{aligned}$$

is valid, because no replacements of x or of y in $e\langle x^+,\, y^-\rangle$ have been replaced. In fact, this final sentence is clearly not valid. ∎

We now prove the lemma

Proof (transitive polarity replacement lemma)

We assume throughout that polarity is with respect to $\prec_1$ and $\prec_2$. We suppose that

$$x \prec_1 y$$

and show that

$$e\langle x^+,\, y^-\rangle \prec_2 e\langle y^+,\, x^-\rangle^n,$$

for every positive integer n. The proof is by induction on n.

Base Case: $n = 1$.

In this case, precisely one replacement is made. The desired result

$$e\langle x^+,\, y^-\rangle \prec_2 e\langle y^+,\, x^-\rangle^1$$

follows from the original polarity replacement lemma.

Inductive Step:

For an arbitrary positive integer k, we assume inductively that

$$e\langle x^+,\ y^-\rangle \prec_2 e\langle y^+,\ x^-\rangle^k$$

and show that

$$e\langle x^+,\ y^-\rangle \prec_2 e\langle y^+,\ x^-\rangle^{k+1}.$$

Observe that $e\langle y^+,\ x^-\rangle^{k+1}$ can be obtained from $e\langle y^+,\ x^-\rangle^k$ by replacing precisely one positive occurrence of x with y or one negative occurrence of y with x. Therefore, by the original polarity replacement lemma, we have

$$e\langle y^+,\ x^-\rangle^k \prec_2 e\langle y^+,\ x^-\rangle^{k+1}.$$

Because our induction hypothesis is that $e\langle x^+,\ y^-\rangle \prec_2 e\langle y^+,\ x^-\rangle^k$, and because we have assumed that $\prec_2$ is transitive, we can conclude that

$$e\langle x^+,\ y^-\rangle \prec_2 e\langle y^+,\ x^-\rangle^{k+1},$$

as we wanted to show. ∎

If $\prec_2$ is transitive, the above lemma allows us to replace one or more occurrences of a variable. If $\prec_2$ is both reflexive and transitive, the following lemma allows us to replace zero, one, or more occurrences.

Lemma (reflexive transitive polarity replacement)

For any binary relations $\prec_1$ and $\prec_2$ and expression $e\langle x^+,\ y^-\rangle$, where $\prec_2$ is both reflexive and transitive, the sentence

$$\begin{array}{l} \textit{if } x \prec_1 y \\ \textit{then } e\langle x^+,\ y^-\rangle \prec_2 e\langle y^+,\ x^-\rangle \end{array}$$

is valid. Here $e\langle y^+,\ x^-\rangle$ is the result of replacing in $e\langle x^+,\ y^-\rangle$ certain positive occurrences of x with y and certain negative occurrences of y with x, where polarity is taken in $e\langle x^+,\ y^-\rangle$ with respect to $\prec_1$ and $\prec_2$.

∎

This lemma, as opposed to the *transitive polarity replacement* lemma, admits the possibility of replacing no occurrences at all of x or y in $e\langle x^+,\ y^-\rangle$.

Example

Suppose our theory includes the theories of both finite sets and integers. Take $e\langle x^+,\ y^-\rangle$ to be the expression

$$e\langle x^+,\ y^-\rangle : \quad card(x^+ \sim y^-) - card(y^- \sim x^+)$$

where $x \sim y$ is the difference between the sets x and y, that is, the set of elements of x that do not belong to y. Take $\prec_1$ to be the subset relation $\subseteq$ and $\prec_2$ to be the weak less-than relation $\leq$. Note that, with respect to $\subseteq$ and $\leq$, both occurrences of x are positive and both occurrences of y are negative in $e\langle x^+,\ y^-\rangle$, as the annotations indicate. Also, $\leq$ is both transitive and reflexive.

Therefore, according to the lemma, the following sentences are valid: the sentence

$$\begin{array}{l} \textit{if } x \subseteq y \\ \textit{then } card(x \sim y) - card(y \sim x) \leq card(y \sim x) - card(x \sim y), \end{array}$$

for which all occurrences of x and y in $e\langle x^+, y^-\rangle$ have been replaced, and the sentence

$$\begin{array}{l} \textit{if } x \subseteq y \\ \textit{then } card(x \sim y) - card(y \sim x) \leq card(x \sim y) - card(y \sim x), \end{array}$$

for which no occurrences of x and y in $e\langle x^+, y^-\rangle$ have been replaced. Of course, other valid sentences can be obtained by replacing some, but not all, of the occurrences of x and y in $e\langle x^+, y^-\rangle$. $\lrcorner$

The proof is straightforward.

Proof (reflexive transitive polarity-replacement lemma)

In the case in which no replacements are made, $e\langle y^+, x^-\rangle$ is identical to $e\langle x^+, y^-\rangle$, and the desired result holds because we have supposed that $\prec_2$ is reflexive. In the case in which one or more replacements are made, the desired result follows from the *transitive polarity replacement* lemma, because we have supposed that $\prec_2$ is also transitive. $\lrcorner$

The following consequence of the polarity replacement lemma will be used most frequently:

Proposition (polarity replacement)

For any binary relation $\prec$ and sentence $\mathcal{P}\langle x^+, y^-\rangle$, the sentence

$$\begin{array}{l} \textit{if } x \prec y \\ \textit{then if } \mathcal{P}\langle x^+, y^-\rangle \\ \qquad \textit{then } \mathcal{P}\langle y^+, x^-\rangle \end{array}$$

is valid. Here $\mathcal{P}\langle y^+, x^-\rangle$ is the result of replacing in $\mathcal{P}\langle x^+, y^-\rangle$ certain positive occurrences of x with y and certain negative occurrences of y with x, where polarity is taken in $\mathcal{P}\langle x^+, y^-\rangle$ with respect to $\prec$. $\lrcorner$

Recall that, when we refer to polarity in a sentence with respect to a single relation $\prec$, we mean polarity with respect to $\prec$ and the *if-then* connective. The proposition allows us to replace occurrences of both x and y in the same sentence and (trivially) admits the possibility that no replacements are made.

The proof is immediate.

Proof

Regarded as a relation, the *if-then* connective is reflexive and transitive. The replaced occurrences of x and y are respectively positive and negative in $\mathcal{P}\langle x^+, y^-\rangle$ with respect to $\prec$ and *if-then*. Therefore the proposition is simply an instance of the *reflexive transitive polarity replacement* lemma, taking $\prec_1$ to be $\prec$, $\prec_2$ to be *if-then*, and $e\langle x^+, y^-\rangle$ to be $\mathcal{P}\langle x^+, y^-\rangle$. $\lrcorner$

Example

Suppose our theory includes the theories of finite sets and integers. Take $\mathcal{P}\langle x^+, y^-\rangle$ to be the sentence

$$\mathcal{P}\langle x^+, y^-\rangle : \quad a < card(x^+ \sim y^-) \textit{ and } card(y^- \sim x^+) \leq b.$$

Take $\prec$ to be the subset relation $\subseteq$. Note that, with respect to $\subseteq$, both occurrences of x are positive and both occurrences of y are negative in $\mathcal{P}\langle x^+, y^-\rangle$, as indicated by the annotations. Therefore, according to the proposition, the following sentences are valid: the sentence

$$\begin{array}{l} \textit{if } x \subseteq y \\ \textit{then if } a < card(x \sim y) \textit{ and } card(y \sim x) \leq b \\ \qquad \textit{then } a < card(x \sim x) \textit{ and } card(y \sim y) \leq b, \end{array}$$

for which one occurrence of x and one occurrence of y in $\mathcal{P}\langle x^+,\ y^-\rangle$ has been replaced, the sentence

$$\begin{array}{l} \textit{if } x \subseteq y \\ \textit{then if } a < card(x \sim y) \textit{ and } card(y \sim x) \leq b \\ \qquad \textit{then } a < card(y \sim y) \textit{ and } card(y \sim y) \leq b, \end{array}$$

for which both occurrences of x in $\mathcal{P}\langle x^+,\ y^-\rangle$ have been replaced, and the sentence

$$\begin{array}{l} \textit{if } x \subseteq y \\ \textit{then if } a < card(x \sim y) \textit{ and } card(y \sim x) \leq b \\ \qquad \textit{then } a < card(y \sim x) \textit{ and } card(x \sim y) \leq b, \end{array}$$

for which both occurrences of x and both occurrences of y in $\mathcal{P}\langle x^+,\ y^-\rangle$ have been replaced. $\lrcorner$

We have now developed the mathematical results on relational polarity we need in order to introduce the special-relations rules. But first, we introduce briskly our basic nonclausal deduction system.

4. NONCLAUSAL DEDUCTION

In this section we present a basic nonclausal deduction system, without any special-relations rules. This system bears some resemblance to those of Murray [82] and Stickel [82]; it is based on the system of Manna and Waldinger [80], but is simplified in several respects:

- The system presented here is a refutation system; it attempts to show that a given set of sentences is unsatisfiable. (The original system operates on a tableau of assertions and goals, and attempts to show that at least one of the goals follows from the assertions.)
- The system is presented with no program synthesis capabilities.
- The mathematical induction principle is omitted.

These simplifications have been made for purely expository purposes: the special-relations rules are compatible with a tableau theorem prover and with the induction principle and are of great use in program synthesis, our primary application.

THE DEDUCED SET

The deduction system we describe operates on a set, called the *deduced* set, of sentences in quantifier-free first-order logic. We attempt to show that a given deduced set is unsatisfiable, i.e., that there is no interpretation under which all the sentences are true.

Theorem proving in a first-order axiomatic theory can be reduced to showing the unsatisfiability of such a set. In particular, to show that a sentence $\mathcal{F}$ is valid in a theory whose axioms are $\mathcal{A}_1,\ \mathcal{A}_2,\ \ldots,\ \mathcal{A}_k$, we can

- Remove the quantifiers of the sentences $\mathcal{A}_1,\ \mathcal{A}_2,\ \ldots,\ \mathcal{A}_k$, and *not* $\mathcal{F}$, by skolemization (see, for example, Chang and Lee [73], Loveland [78], or Robinson [79]).
- Show the unsatisfiability of the resulting set of quantifier-free sentences.

We do not require that the sentences be in clausal form; indeed, they can use the full set of connectives of propositional logic, including equivalence ($\equiv$) and the conditional (*if-then-else*).

Example

Consider the theory of the *strict partial ordering* $\prec$, defined by the *transitivity* axiom

$$(\forall x)(\forall y)(\forall z)\begin{bmatrix} \textit{if}\ x \prec y\ \textit{and}\ y \prec z \\ \textit{then}\ x \prec z \end{bmatrix}$$

and the *irreflexivity* axiom

$$(\forall x)\big[\textit{not}\,(x \prec x)\big].$$

Suppose we would like to show that in this theory the *asymmetry* property

$$(\forall u)(\forall v)\begin{bmatrix} \textit{if}\ u \prec v \\ \textit{then}\ \ \textit{not}\ v \prec u \end{bmatrix}$$

is valid. It suffices to show that the set of quantifier-free sentences

$$\begin{matrix} \textit{if}\ x \prec y\ \textit{and}\ y \prec z \\ \textit{then}\ x \prec z \end{matrix} \qquad \textit{not}\,(x \prec x) \qquad \textit{not}\begin{bmatrix} \textit{if}\ a \prec b \\ \textit{then}\ \ \textit{not}\,(b \prec a) \end{bmatrix}$$

is unsatisfiable. ▟

If the truth symbol *false* belongs to the deduced set, the set is automatically unsatisfiable, because the sentence *false* is not true under any interpretation.

Because the variables of the sentences in the deduced set are tacitly quantified universally, we can systematically rename them without changing the unsatisfiability of the set; that is, the set is unsatisfiable before the renaming if and only if it is unsatisfiable afterwards. Of course, we must replace every occurrence of a variable in the sentence with the new variable, and we must be careful not to replace distinct variables in the sentence with the same variable. The variables of the sentences in the deduced set may therefore be *standardized apart*; in other words, we may rename the variables of the sentences so that no two of them have variables in common.

For any sentence $\mathcal{F}$ in the deduced set and any substitution θ, we may add to the set the *instance* $\mathcal{F}\theta$ of $\mathcal{F}$, without changing the unsatisfiability of the set. In particular, if the deduced set is unsatisfiable after the addition of the new sentence, it was also unsatisfiable before. Note that in adding the new sentence $\mathcal{F}\theta$, we do not remove the original sentence $\mathcal{F}$.

THE DEDUCTIVE PROCESS

In the deductive system we apply *deduction rules*, which add new sentences to the deduced set without changing its unsatisfiability. Deduction rules are expressed as follows:

$$\frac{\mathcal{F}_1,\ \mathcal{F}_2,\ \ldots,\ \mathcal{F}_m}{\mathcal{F}}$$

This means that, if the *given* sentences $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_m$ belong to the deduced set, the *conclusion* $\mathcal{F}$ may be added. Such a rule is said to be *sound* if the given sentences $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_m$ imply the sentence $\mathcal{F}$. If a deductive rule is sound, its application will preserve the unsatisfiability of the deduced set.

The deductive process terminates successfully if we introduce the truth symbol *false* into the deduced set. Because deduction rules preserve unsatisfiability, and because a set of sentences containing *false* is automatically unsatisfiable, this will imply that the original deduced set was also unsatisfiable.

We include two classes of deduction rules in the basic system:

- The *transformation* rules, which replace subsentences with equivalent sentences.
- The *resolution* rule, which performs a case analysis on the truth of matching subsentences.

These rules are described in this section. In later sections, we augment the basic system with two new classes of rules:

- The *replacement* rules, which replace subexpressions with other expressions (not necessarily equivalent or equal).
- The *matching* rules, which introduce new conditions to be proved that enable subexpressions to be matched.

We first describe the transformation rules.

TRANSFORMATION RULES

The transformation rules replace subsentences of the sentences of our deduced set with propositionally equivalent, simpler sentences. For instance, the transformation rule

$$\mathcal{P}\ \textit{and true} \rightarrow \mathcal{P}$$

replaces a subsentence of form ($\mathcal{P}$ *and true*) with the corresponding sentence of form $\mathcal{P}$. The simplified sentence is then added to the deduced set. (Logically speaking, the original sentence remains in the deduced set too, but, for efficiency of implementation, the original sentence need not be retained.)

We include a full set of such *true-false* transformation rules; e.g.,

$$\textit{not true} \rightarrow \textit{false}$$

$$\mathcal{P}\ \textit{or true} \rightarrow \textit{true}$$

$$\textit{if}\ \mathcal{P}\ \textit{then false} \rightarrow \textit{not}\ \mathcal{P}.$$

These rules can eliminate from a sentence any occurrence of the truth symbols *true* and *false* as a proper subsentence.

We also include such propositional simplification rules as

$$\mathcal{P}\ \textit{and}\ \mathcal{P} \rightarrow \mathcal{P}$$

$$\textit{not not}\ \mathcal{P} \rightarrow \mathcal{P}.$$

These rules are not logically necessary, but are included for cosmetic purposes.

The soundness of the transformation rules is evident, because each produces a sentence equivalent to the one to which it is applied.

Example

Suppose our deduced set contains the sentence

$$\mathcal{F}: \begin{array}{l} \textit{if}\ q(a)\ \textit{then false} \\ \quad \textit{or} \\ (\textit{not true})\ \ \textit{or}\ \ \big(\textit{not}\ q(a)\big). \end{array}$$

(We omit parentheses when the structure of the sentence can be indicated by indenting.) This can be transformed, by application of the rule

$$\textit{if}\ \mathcal{P}\ \textit{then false} \rightarrow \textit{not}\ \mathcal{P},$$

into the sentence

$$\begin{array}{l} \textit{not}\ q(a) \\ \quad \textit{or} \\ (\textit{not true})\ \ \textit{or}\ \ (\textit{not}\ q(a)), \end{array}$$

which may then be added to the deduced set.

The new sentence can be transformed in turn, by successive application of the rules

$$\textit{not true} \rightarrow \textit{false}$$

$$\textit{false}\ \ \textit{or}\ \ \mathcal{P} \rightarrow \mathcal{P},$$

$$\mathcal{P}\ \ \textit{or}\ \ \mathcal{P} \rightarrow \mathcal{P},$$

into the sentence

$$\textit{not}\ q(a).$$

We shall say that the original sentence $\mathcal{F}$ *reduces to* $(\textit{not}\ q(a))$ under transformation. ⌟

Our original system (Manna and Waldinger [80]) included many more transformation rules; also, their operation was more complex. In this system, the role of these more complex rules has been assumed by the replacement rule of Section 5.

RESOLUTION RULE: GROUND VERSION

The resolution rule applies to two sentences of our set, and performs a case analysis on the truth of a common subsentence. Instances of the sentences can be formed, if necessary, to create a common subsentence; however, we first present the *ground version* of the rule, which does not form instances of these sentences.

Rule (resolution, ground version)

For any ground sentences $\mathcal{P}$, $\mathcal{F}[\mathcal{P}]$, and $\mathcal{G}[\mathcal{P}]$, we have

$$\frac{\begin{array}{c}\mathcal{F}[\mathcal{P}] \\ \mathcal{G}[\mathcal{P}]\end{array}}{\mathcal{F}[\textit{false}]\ \ \textit{or}\ \ \mathcal{G}[\textit{true}]} \quad ⌟$$

In other words, if $\mathcal{F}[\mathcal{P}]$ and $\mathcal{G}[\mathcal{P}]$ are sentences in our deduced set with a common subsentence $\mathcal{P}$, we can add to the set the sentence $(\mathcal{F}[\textit{false}]\ \textit{or}\ \mathcal{G}[\textit{true}])$ obtained by replacing every occurrence of $\mathcal{P}$ in $\mathcal{F}[\mathcal{P}]$ with *false*, replacing every occurrence of $\mathcal{P}$ in $\mathcal{G}[\mathcal{P}]$ with *true*, and taking the disjunction of the results. We shall assume that $\mathcal{F}[\mathcal{P}]$ and $\mathcal{G}[\mathcal{P}]$ have at least one occurrence each of the subsentence $\mathcal{P}$. We do not require that $\mathcal{F}[\mathcal{P}]$ and $\mathcal{G}[\mathcal{P}]$ be distinct sentences.

Because the resolution rule introduces new occurrences of the truth symbols *true* and *false*, it is always possible to simplify the resulting sentence immediately afterwards by application of the appropriate *true-false* rules. These subsequent transformations will sometimes be regarded as part of the resolution rule itself.

Example

Suppose our deduced set contains the sentences

$$\mathcal{F}: \quad \textit{if } q(a) \textit{ then } \boxed{p(a,\, b)}$$

and

$$\mathcal{G}: \quad (\textit{not} \boxed{p(a,\, b)}) \textit{ or } (\textit{not}\, q(a)).$$

These sentences have a common subsentence $p(a,\, b)$, indicated by the surrounding boxes. By application of the resolution rule, we may replace every occurrence of $p(a,\, b)$ in $\mathcal{F}$ with *false*, replace every occurrence of $p(a,\, b)$ in $\mathcal{G}$ with *true*, and take the disjunction of the result, obtaining the sentence

$$\begin{array}{l} \textit{if } q(a) \textit{ then false} \\ \quad \textit{or} \\ (\textit{not true}) \textit{ or } (\textit{not}\, q(a)), \end{array}$$

which (as we have seen in a previous example) reduces under transformation to

$$\textit{not}\, q(a).$$

This sentence may be added to the deduced set. $\lrcorner$

Let us show that the resolution rule is sound, and hence that it preserves the unsatisfiability of the deduced set.

Justification (resolution rule, ground version)

We must show that the given sentences $\mathcal{F}[\mathcal{P}]$ and $\mathcal{G}[\mathcal{P}]$ imply the newly deduced sentence $(\mathcal{F}[\textit{false}] \textit{ or } \mathcal{G}[\textit{true}])$. Suppose that $\mathcal{F}[\mathcal{P}]$ and $\mathcal{G}[\mathcal{P}]$ are true; we would like to show that then $(\mathcal{F}[\textit{false}] \textit{ or } \mathcal{G}[\textit{true}])$ is true. We show that one of the two disjuncts, $\mathcal{F}[\textit{false}]$ or $\mathcal{G}[\textit{true}]$, is true.

In the case in which the common subsentence $\mathcal{P}$ is false, we know (by the ***value*** property, because $\mathcal{P}$ and *false* have the same truth value and $\mathcal{F}[\mathcal{P}]$ is true) that the first of the disjuncts, $\mathcal{F}[\textit{false}]$, is true.

Similarly, in the case in which the common subsentence $\mathcal{P}$ is true, we know (by the ***value*** property again, because $\mathcal{P}$ and *true* have the same truth value and $\mathcal{G}[\mathcal{P}]$ is true) that the second of the disjuncts, $\mathcal{G}[\textit{true}]$, is true. $\lrcorner$

We have established the soundness of the ground version of the resolution rule when applied to ground sentences, which contain no variables. We require the sentences to be ground because the justification depends on the *value* property, which holds only for ground sentences. We can actually apply the ground version of the rule to sentences with variables; the soundnes of such applications follows from the justification for the general version of the rule, which we present later.

We now discuss an important strategy for controlling the resolution rule.

THE POLARITY STRATEGY

Murray's [82] *polarity strategy* allows us to consider only those applications of the resolution rule under which at least one occurrence of $\mathcal{P}$ is positive (or of no polarity) in $\mathcal{F}[\mathcal{P}]$ and at least one occurrence of $\mathcal{P}$ is negative (or of no polarity) in $\mathcal{G}[\mathcal{P}]$. In other words, not all the subsentences that are replaced with *false* are negative and not all the subsentences that are replaced with *true* are positive. This strategy blocks many useless applications of the rule and rarely interferes with a reasonable step.

The intuitive rationale for the polarity strategy is that it is our goal to deduce the sentence *false*, which is more false than any other sentence. By replacing positive sentences with *false* and negative sentences with *true*, we are moving in the right direction, making the entire sentence more false.

Example

Suppose our deduced set contains the sentences

$$\mathcal{F}:\quad \boxed{p(a)}^{+}\ \textit{ or }\ q(b)$$

and

$$\mathcal{G}:\quad \textit{if }\boxed{p(a)}^{-}\ \textit{ then }\ q(b).$$

These sentences have occurrences of a common subsentence $p(a)$, of positive and negative polarity, respectively, as indicated by the annotation. By application of the resolution rule, we obtain the sentence

$$\begin{array}{l}\textit{false or } q(b)\\ \quad \textit{or}\\ \textit{if true then } q(b),\end{array}$$

which reduces to $q(b)$ under transformation.

Let us reverse the roles of our sentences.

$$\mathcal{F}:\quad \textit{if }\boxed{p(a)}^{-}\ \textit{ then }\ q(b)$$

$$\mathcal{G}:\quad \boxed{p(a)}^{+}\ \textit{ or }\ q(b).$$

The sentences still have occurrences of a common subsentence $p(a)$. However, it is in violation of the polarity strategy to apply the rule for the sentences in this order, because now the occurrence of $p(a)$ is negative in $\mathcal{F}$, i.e., it is not positive or of no polarity. Also, the polarity of $p(a)$ is positive in $\mathcal{G}$. If we insist on applying the resolution rule anyway, we obtain the sentence

$$\begin{array}{l}\textit{if false then } q(b)\\ \quad \textit{or}\\ \textit{true or } q(b),\end{array}$$

which reduces to *true* under transformation. Although it does no harm to add the sentence *true* to our deduced set, it is of no use in establishing the unsatisfiability of the set.

There are two other legal applications of the resolution rule to the same two sentences, obtained by taking the common subsentence to be $q(b)$ rather than $p(a)$. Both of these applications of the rule lead us to obtain the redundant sentence *true*, and both are in violation of the polarity strategy. ⌟

RESOLUTION RULE: GENERAL VERSION

The general version of the rule allows us to instantiate the variables of the given sentences as necessary to create common subsentences. It is expressed as follows:

Rule (resolution, general version)

For any sentences $\mathcal{P}$, $\tilde{\mathcal{P}}$, $\mathcal{F}[\mathcal{P}]$, and $\mathcal{G}[\tilde{\mathcal{P}}]$, where $\mathcal{F}$ and $\mathcal{G}$ are standardized apart, i.e., they have no variables in common, we have

$$\begin{array}{c} \mathcal{F}[\mathcal{P}] \\ \mathcal{G}[\tilde{\mathcal{P}}] \\ \hline \mathcal{F}\theta[\mathit{false}] \ \mathit{or} \ \mathcal{G}\theta[\mathit{true}] \end{array}$$

where θ is a most-general unifier of $\mathcal{P}$ and $\tilde{\mathcal{P}}$.

More precisely,

- $\mathcal{F}$ has one or more subsentences $\mathcal{P}$, $\mathcal{P}_1$, $\mathcal{P}_2$, $\ldots$.
- $\mathcal{G}$ has one or more subsentences $\tilde{\mathcal{P}}$, $\tilde{\mathcal{P}}_1$, $\tilde{\mathcal{P}}_2$, $\ldots$.
- θ is a most general unifier of $\mathcal{P}$, $\mathcal{P}_1$, $\mathcal{P}_2$, $\ldots$, and $\tilde{\mathcal{P}}$, $\tilde{\mathcal{P}}_1$, $\tilde{\mathcal{P}}_2$, $\ldots$; hence

  $$\mathcal{P}\theta = \mathcal{P}_1\theta = \mathcal{P}_2\theta = \ldots = \tilde{\mathcal{P}}\theta = \tilde{\mathcal{P}}_1\theta = \tilde{\mathcal{P}}_2\theta = \ldots .$$
- The conclusion of the rule is obtained by replacing all occurrences of $\mathcal{P}\theta$ in $\mathcal{F}\theta$ with *false*, replacing all occurrences of $\tilde{\mathcal{P}}\theta$ (that is, $\mathcal{P}\theta$) in $\mathcal{G}\theta$ with ***true***, and taking the disjunction of the results

In other words, we apply the ground version of the rule to $\mathcal{F}\theta$ and $\mathcal{G}\theta$, taking $\mathcal{P}\theta$ as the common subsentence. ⌟

The rule requires that the sentences $\mathcal{F}$ and $\mathcal{G}$ be standardized apart, i.e., that they have no variables in common. This may be achieved by renaming the variables of the sentences as necessary. If both are the same sentence, we rename the variables of one copy of the sentence.

Let us show that the general version of the rule is sound.

Justification (resolution rule, general version):

The soundness of the general version of the rule follows from the soundness of its ground version. We show that the sentences $\mathcal{F}$ and $\mathcal{G}$ imply the sentence $(\mathcal{F}\theta[\mathit{false}] \ \mathit{or} \ \mathcal{G}\theta[\mathbf{\mathit{true}}])$.

We suppose that [under a given interpretation] the sentences $\mathcal{F}$ and $\mathcal{G}$ are true and show that $(\mathcal{F}\theta[\mathit{false}] \ \mathit{or} \ \mathcal{G}\theta[\mathbf{\mathit{true}}])$ is also true. It suffices (by the definition of truth for a nonground sentence) to show that any ground instance of $(\mathcal{F}\theta[\mathit{false}] \ \mathit{or} \ \mathcal{G}\theta[\mathbf{\mathit{true}}])$ is true.

Because $\mathcal{F}$ and $\mathcal{G}$ are true, we know (by the *instantiation* lemma) that $\mathcal{F}\theta$ and $\mathcal{G}\theta$ are true and hence (by the definition of truth for a nonground sentence) that every ground instance of $\mathcal{F}\theta$ and $\mathcal{G}\theta$ is true. But any ground instance of $(\mathcal{F}\theta[\mathit{false}] \ \mathit{or} \ \mathcal{G}\theta[\mathit{true}])$ is the result of applying the ground version of the rule to the corresponding ground instance of $\mathcal{F}\theta$ and $\mathcal{G}\theta$; therefore it is also true. ⌟

The general version of the rule includes the ground version as a special case, in which the most-general unifier θ is the empty substitution $\{\ \}$.

The following illustration of the general resolution rule is extracted from the derivation of a binary-search real-number square-root program.

Example

In the theory of the nonnegative real numbers, suppose our deduced set contains the sentence

$$\mathcal{F}: \quad \textit{not}\,\big(y^2 \le a \ \ \textit{and}\ \ \textit{not}\,\boxed{(y+\epsilon)^2 \le a}^{+}\big),$$

where y is a variable and a and ϵ are constants. (The sentence is negated because it is deduced from the negation of the original theorem.)

We are about to apply the resolution rule to this sentence and itself. Therefore let us produce another copy of the sentence and standardize the two sentences apart; i.e., we rename the variable of the second sentence

$$\mathcal{G}: \quad \textit{not}\,\big(\boxed{\tilde{y}^2 \le a}^{-} \ \ \textit{and}\ \ \textit{not}\,((\tilde{y}+\epsilon)^2 \le a)\big).$$

The boxed subsentences

$$\mathcal{P}: \quad (y+\epsilon)^2 \le a$$

and

$$\tilde{\mathcal{P}}: \quad \tilde{y}^2 \le a$$

are unifiable, with most-general unifier

$$\theta: \quad \{\tilde{y} \leftarrow y+\epsilon\}.$$

To apply the rule, we replace all occurrences of $\mathcal{P}\theta$ in $\mathcal{F}\theta$ with *false*, replace all occurrences of $\tilde{\mathcal{P}}\theta$ in $\mathcal{G}\theta$ with *true*, and take the disjunction of the results, obtaining

$$\begin{array}{l} \textit{not}\,(y^2 \le a \ \ \textit{and}\ \ \textit{not false}) \\ \quad \textit{or} \\ \textit{not}\,\big(\textit{true}\ \ \textit{and}\ \ \textit{not}\,(((y+\epsilon)+\epsilon)^2 \le a)\big). \end{array}$$

This sentence reduces under transformation to

$$\textit{not}\,(y^2 \le a) \ \ \textit{or}\ \ \big((y+\epsilon)+\epsilon\big)^2 \le a.$$

The above application of the rule is in accordance with the polarity strategy, because the boxed subsentence $\mathcal{P}$ is positive in $\mathcal{F}$ and the boxed subsentence $\tilde{\mathcal{P}}$ is negative in $\mathcal{G}$. ┘

The resolution rule presented here is an extension of the rule of Robinson [65] to the nonclausal case. Robinson's rule applies to clauses of the form

$$\mathcal{F}: \quad \mathcal{P} \ \ \textit{or}\ \ \mathcal{F}'$$

$$\mathcal{G}: \quad (\textit{not}\ \tilde{\mathcal{P}}) \ \ \textit{or}\ \ \mathcal{G}',$$

where $\mathcal{P}$ and $\tilde{\mathcal{P}}$ are unifiable propositions, with most-general unifier θ, and $\mathcal{F}'$ and $\mathcal{G}'$ are themselves clauses. Robinson's rule deduces the new sentence

$$\mathcal{F}'\theta \ \ \textit{or}\ \ \mathcal{G}'\theta.$$

The resolution rule presented here deduces, from the same sentences $\mathcal{F}$ and $\mathcal{G}$, the new sentence

$$\begin{array}{l} \textit{false}\ \ \textit{or}\ \ \mathcal{F}'\theta \\ \quad \textit{or} \\ (\textit{not true})\ \ \textit{or}\ \ \mathcal{G}'\theta. \end{array}$$

This sentence reduces under transformation to ($\mathcal{F}'\theta$ *or* $\mathcal{G}'\theta$), the same sentence deduced by Robinson's version of the rule.

Nonclausal resolution was developed independently by Manna and Waldinger [80] and Murray [82]. The resolution and transformation rules together have been shown by Murray to provide a complete system for first-order logic. An implementation of a nonclausal resolution theorem prover by Stickel [82] employs a connection graph strategy.

5. THE RELATION REPLACEMENT RULE

We now begin to extend our nonclausal deduction system to give special treatment to a binary relation $\prec$. The two new rules of the extension allow us to build into the system instances of the *polarity replacement* proposition, just as the paramodulation and E-resolution rules allow us to build in instances of the substitutivity of equality.

Recall that, according to the *polarity replacement* proposition, for any sentence $\mathcal{P}\langle x^+,\ y^-\rangle$ and binary relation $\prec$, the sentence

$$\begin{array}{l} \textit{if } x \prec y \\ \textit{then if } \mathcal{P}\langle x^+,\ y^-\rangle \textit{ then } \mathcal{P}\langle y^+,\ x^-\rangle \end{array}$$

is valid.

If we could add this sentence to our deduced set for each relevant sentence $\mathcal{P}\langle x^+, y^-\rangle$, we could achieve a considerable abbreviation of the proof, at the cost of a dramatic explosion of the search space. The extended system will behave as if the sentences were present, achieving the same abbreviation of the proof and, at the same time, collapsing rather than exploding the search space.

We begin with the relation replacement rule, which is our generalization of the paramodulation rule.

THE GROUND VERSION

With respect to a given relation $\prec$, the rule allows us to replace subexpression occurrences with larger or smaller expressions, depending on their polarity. The ground version of the rule which applies to sentences with no variables, is as follows:

Rule (relation replacement, ground version)

For any binary relation $\prec$, ground expressions s and t, and ground sentences $\mathcal{F}[s \prec t]$ and $\mathcal{G}\langle s^+, t^-\rangle$, we have

$$\frac{\begin{array}{c}\mathcal{F}[s \prec t] \\ \mathcal{G}\langle s^+,\ t^-\rangle\end{array}}{\mathcal{F}[\mathit{false}] \ \textit{or}\ \mathcal{G}\langle t^+,\ s^-\rangle.}$$

Here $\mathcal{G}\langle t^+,\ s^-\rangle$ is obtained from $\mathcal{G}\langle s^+,\ t^-\rangle$ by replacing certain positive occurrences of s with t and replacing certain negative occurrences of t with s, where polarity is taken in $\mathcal{G}\langle s^+,\ t^-\rangle$ with respect to $\prec$. ⌟

In other words, if $\mathcal{F}[s \prec t]$ and $\mathcal{G}\langle s^+, t^-\rangle$ are sentences in our deduced set, we can add to the set the sentence $(\mathcal{F}[\mathit{false}]\ \textit{or}\ \mathcal{G}\langle t^+,\ s^-\rangle)$.

For a particular relation $\prec$, we shall refer to this rule as the $\prec$-replacement rule: thus, we have a $<$-replacement rule, a $\leq$-replacement rule, and so forth. Although the rule allows us to replace occurrences in

$\mathcal{G}\langle s^+, t^-\rangle$ of both expressions s and t at the same time, it is typically applied to replace occurrences of one or the other expression, but not both. Subsequent application of transformation rules, to remove occurrences of the truth symbols *true* and *false*, may be regarded as part of the relation replacement rule itself.

There is a polarity strategy for the relation replacement rule, which allows us to apply the rule only if some occurrence of $s \prec t$ is positive (or of no polarity) in $\mathcal{F}[s \prec t]$.

Naturally we may also require that some occurrence of s or t is actually replaced; otherwise, $\mathcal{G}\langle t^+, s^-\rangle$ is identical to $\mathcal{G}\langle s^+, t^-\rangle$, and the sentence we obtain is $(\mathcal{F}[\textit{false}]\ \textit{or}\ \mathcal{G}\langle s^+, t^-\rangle)$; this is weaker than the sentence $\mathcal{G}\langle s^+, t^-\rangle$, which was already in the deduced set.

In illustrating the rule we draw boxes around the matching occurrences of s and t.

Example

In the theory of the nonnegative integers, suppose our deduced set contains the sentences

$$\mathcal{F}: \quad \begin{array}{l} \textit{if}\ p(s) \\ \textit{then}\ (\boxed{s} < t)^+ \end{array}$$

and

$$\mathcal{G}: \quad s < \boxed{s^+}^2$$

Note that the boxed occurrence of s in $\mathcal{G}$ is positive with respect to the less-than relation $<$. Therefore we can apply the $<$-replacement rule to replace the occurrence of s in $\mathcal{G}$ with t, to deduce

$$\begin{bmatrix} \textit{if}\ p(s) \\ \textit{then}\ \textit{false} \end{bmatrix} \quad \textit{or}\quad s < t^2,$$

which reduces under transformation to

$$(\textit{not}\, p(s)) \quad \textit{or} \quad s < t^2.$$

The above application of the rule is in accordance with the polarity strategy, because the occurrence of $s < t$ is positive in $\mathcal{F}$. Note that not every occurrence of s in $\mathcal{G}$ was replaced in applying the rule.

In a system without the relation replacement rule, we could have deduced the same conclusion by applying the resolution rule in sequence to $\mathcal{F}$, $\mathcal{G}$, the *monotonicity* property

$$\begin{array}{l} \textit{if}\ x < y \\ \textit{then}\ x^2 < y^2, \end{array}$$

and the *transitivity* property

$$\begin{array}{l} \textit{if}\ x < y \\ \textit{then}\ \textit{if}\ y < z \\ \qquad \textit{then}\ x < z. \end{array}$$

The rule allows us to draw the conclusion even if the *monotonicity* and *transitivity* properties are not in our deduced set. ┘

The following illustration of the rule is extracted from the derivation of a program to find the maximum element of a list of numbers.

Example

In a theory of lists of numbers (integers, say), suppose our deduced set contains the sentences

$$\mathcal{F}: \quad \begin{array}{l} not \begin{bmatrix} \mathit{if}\ g(m) = h \\ \mathit{then}\ \mathit{not}\ (m < \boxed{h})^{+} \end{bmatrix} \\ or \\ t = [\ ] \end{array}$$

and

$$\mathcal{G}: \quad not \begin{bmatrix} \mathit{if}\ g(h) \in t \\ \mathit{then}\ g(h) \leq \boxed{h}^{-} \end{bmatrix}.$$

Note that the boxed occurrence of h in $\mathcal{G}$ is negative with respect to $<$. Therefore we can apply the $<$-replacement rule to replace the occurrence of h in $\mathcal{G}$ with m, to deduce

$$\begin{array}{l} \begin{bmatrix} not \begin{bmatrix} \mathit{if}\ g(m) = h \\ \mathit{then}\ \mathit{not}\ \mathit{false} \end{bmatrix} \\ \quad or \\ t = [\ ] \end{bmatrix} \\ \quad or \\ not \begin{bmatrix} \mathit{if}\ g(h) \in t \\ \mathit{then}\ g(h) \leq m) \end{bmatrix}. \end{array}$$

This sentence reduces under *true-false* transformation to

$$\begin{array}{l} t = [\ ] \\ \quad or \\ not \begin{bmatrix} \mathit{if}\ g(h) \in t \\ \mathit{then}\ g(h) \leq m \end{bmatrix}. \end{array}$$

The above application of the rule is in accordance with the polarity strategy, because the subsentence $m < h$ is positive in $\mathcal{F}$. ⌟

Let us now establish the soundness of the rule.

Justification (relation replacement, ground version)

We show that the given sentences $\mathcal{F}[s \prec t]$ and $\mathcal{G}\langle s^{+},\ t^{-}\rangle$ imply the conclusion $(\mathcal{F}[\mathit{false}]\ \mathit{or}\ \mathcal{G}\langle t^{+},\ s^{-}\rangle)$. We distinguish between two cases and show that in each case one of the two disjuncts, $\mathcal{F}[\mathit{false}]$ or $\mathcal{G}\langle t^{+},\ s^{-}\rangle$, is true.

In the case in which the subsentence $s \prec t$ is false, we know (by the *value* property, because $s \prec t$ and *false* have the same truth value and $\mathcal{F}[s \prec t]$ is true) that the first of the disjuncts, $\mathcal{F}[\mathit{false}]$, is true.

In the case in which $s \prec t$ is true, we know (by the *polarity replacement* proposition, because $\mathcal{G}\langle s^{+},\ t^{-}\rangle$ is true) that the second of the disjuncts, $\mathcal{G}\langle t^{+},\ s^{-}\rangle$, is true. ⌟

As with the resolution rule, we have established the soundness of the ground version of the relation replacement rule when applied to sentences with no variables. We will actually apply the ground version of the rule to sentences with variables. The above justification does not extend to this case, however, because the *value* property only holds for ground sentences. Such applications are an instance of the following general version of the rule.

THE GENERAL VERSION

We are now ready to give the general version of the rule, which applies to sentences with variables and allows us to instantiate the variables as necessary to create common subexpressions.

Rule (relation replacement, general version)

For any binary relation $\prec$, expressions s, t, $\widetilde{s}$, and $\widetilde{t}$, and sentences $\mathcal{F}[s \prec t]$ and $\mathcal{G}\langle \widetilde{s}^{+}, \widetilde{t}^{-}\rangle$, where $\mathcal{F}$ and $\mathcal{G}$ are standardized apart, we have

$$\frac{\begin{array}{c}\mathcal{F}[s \prec t] \\ \mathcal{G}\langle \widetilde{s}^{+}, \widetilde{t}^{-}\rangle\end{array}}{\mathcal{F}\theta[\mathit{false}] \ \ \mathit{or} \ \ \mathcal{G}\theta\langle t\theta^{+}, s\theta^{-}\rangle}$$

where θ is a simultaneous, most-general unifier of $s, \widetilde{s}$ and of $t, \widetilde{t}$.

More precisely,

- $\mathcal{F}$ has one or more subsentences $s \prec t$, $s_1 \prec t_1$, $s_2 \prec t_2$, ...
- $\mathcal{G}$ has one or more subexpressions $\widetilde{s}, \widetilde{s}_1, \widetilde{s}_2, \ldots$ and $\widetilde{t}, \widetilde{t}_1, \widetilde{t}_2, \ldots$.
- θ is a simultaneous most-general unifier of $s, s_1, s_2, \ldots, \widetilde{s}, \widetilde{s}_1, \widetilde{s}_2, \ldots$ and of $t, t_1, t_2, \ldots, \widetilde{t}, \widetilde{t}_1, \widetilde{t}_2, \ldots$; hence

  $$s\theta = s_1\theta = s_2\theta = \ \ldots \ = \widetilde{s}\theta = \widetilde{s}_1\theta = \widetilde{s}_2\theta = \ldots$$

  and

  $$t\theta = t_1\theta = t_2\theta = \ \ldots \ = \widetilde{t}\theta = \widetilde{t}_1\theta = \widetilde{t}_2\theta = \ \ldots.$$

- The conclusion of the rule is obtained by replacing all occurrences of $(s \prec t)\theta$ in $\mathcal{F}\theta$ with *false*, replacing certain positive occurrences of $s\theta$ in $\mathcal{G}\theta$ with $t\theta$, replacing certain negative occurrences of $t\theta$ in $\mathcal{G}\theta$ with $s\theta$, and taking the disjunction of the two results. Here polarity is in $\mathcal{G}\theta$ with respect to $\prec$.

In other words, we apply the ground version of the rule to $\mathcal{F}\theta$ and $\mathcal{G}\theta$. ┘

The justification of the general version of the rule, which we omit, is straightforward now that the soundness of the ground version has been established. The proof is analogous to the proof of the general version of the resolution rule. The polarity strategy for this rule allows us to assume that at least one occurrence of the subsentence $(s \prec t)\theta$ is positive or of no polarity in $\mathcal{F}\theta$.

Example

In the theory of sets, suppose our deduced set contains the sentences

$$\mathcal{F}: \quad \begin{array}{l}\mathit{if}\ p(x) \\ \mathit{then}\ \left(\boxed{h(x,a)} \subset \boxed{b}\right)^{+} \ \mathit{or}\ \left(\boxed{h(b,y)} \subset \boxed{x}\right)^{+}\end{array}$$

and

$$\mathcal{G}: \quad \left(c \in \boxed{h(u,a)}^{+} \sim v\right) \ \mathit{or}\ q(u,v),$$

where $\sim$ is the set difference function.

Note that

- $\mathcal{F}$ contains the [positive] subsentences $h(x, a) \subset b$ and $h(b, y) \subset x$.
- The boxed subterms $h(x, a), h(b, y)$, and $h(u, a)$ and the boxed subterms b and x are simultaneously unifiable, with most-general unifier

  $$\theta : \quad \{x \leftarrow b,\ u \leftarrow b,\ y \leftarrow a\}.$$

- The boxed occurrence of $h(u, a)$ is positive in $\mathcal{G}$ with respect to $\subset$.

Therefore we can apply the $\subset$-replacement rule, replacing all occurrences of $h(b, a) \subset b$ in $\mathcal{F}\theta$ with *false*, replacing the occurrence of $h(b, a)$ in $\mathcal{G}\theta$ with b, and taking the disjunction of the results, to obtain

$$\begin{array}{l} \begin{bmatrix} \textit{if } p(b) \\ \textit{then false or false} \end{bmatrix} \\ \quad \textit{or} \\ (c \in b \sim v) \;\; \textit{or} \;\; q(b, v). \end{array}$$

This sentence reduces under transformation to

$$\big(\textit{not}\, p(b)\big) \;\; \textit{or} \;\; (c \in b \sim v) \;\; \textit{or} \;\; q(b, v).$$

The above application of the rule is in accordance with the polarity strategy. ■

Use of the relation replacement rule allows a dramatic abbreviation of many proofs. For this reason and because the rule enables us to eliminate troublesome axioms from the deduced set, the search space is constricted. We have not established completeness results for the rule; judging from the corresponding theorem for paramodulation (Brand [75]), we expect such results to be difficult.

SPECIAL CASE: THE EQUALITY REPLACEMENT RULE

The most important instance of the relation replacement rule is obtained by taking the relation $\prec$ to be the equality relation $=$. This special case of the rule, which allows us to replace equals with equals, is a nonclausal version of the paramodulation rule. It may be expressed as follows:

Rule (equality replacement)

For any terms s, t, $\tilde{s}$, and $\tilde{t}$, and sentences $\mathcal{F}[s = t]$ and $\mathcal{G}\langle \tilde{s},\ \tilde{t}\rangle$, where $\mathcal{F}$ and $\mathcal{G}$ are standardized apart, we have

$$\begin{array}{c} \mathcal{F}[s = t] \\ \mathcal{G}\langle \tilde{s},\ \tilde{t}\rangle \\ \hline \mathcal{F}\theta[\textit{false}] \;\; \textit{or} \;\; \mathcal{G}\theta\langle t\theta, s\theta\rangle \end{array}$$

where θ is a simultaneous, most-general unifier of s, $\tilde{s}$ and of t, $\tilde{t}$. ■

The notation is analogous to that for the general relation-replacement rule. We do not need to restrict the polarity of the replaced subterms $s\theta$ and $t\theta$ in $\mathcal{G}\theta$, because any term has both polarities with respect to the equality relation. The polarity strategy is the same as before.

The following illustration of the equality replacement rule is extracted from the derivation of an integer quotient program.

Example

In the theory of the nonnegative integers, suppose our deduced set contains the sentences

$$\mathcal{F}: \quad (\boxed{0 \cdot u} = 0)^{+}$$

and

$$\mathcal{G}: \quad \mathit{not}\left(\boxed{z \cdot d} \leq n \;\; \mathit{and} \;\; (z+1) \cdot d > n\right).$$

(In the derivation, $\mathcal{F}$ is an axiom and $\mathcal{G}$ is deduced from the negation of the theorem.)

Note that

- $\mathcal{F}$ contains the (positive) subsentence $0 \cdot u = 0$.
- The boxed subterms $0 \cdot u$ and $z \cdot d$ are unifiable, with most-general unifier

$$\theta: \quad \{z \leftarrow 0, \; u \leftarrow d\}.$$

Therefore we can apply the =-replacement rule, replacing all occurrences of $0 \cdot d = 0$ in $\mathcal{F}\theta$ with *false*, replacing the occurrence of $0 \cdot d$ in $\mathcal{G}\theta$ with 0, and taking the disjunction of the results, to deduce

$$\begin{array}{l} \mathit{false} \\ \quad \mathit{or} \\ \mathit{not}\left(0 \leq n \;\; \mathit{and} \;\; (0+1) \cdot d > n\right). \end{array}$$

This sentence reduces under *true-false* transformation to

$$\mathit{not}\left(0 \leq n \;\; \mathit{and} \;\; (0+1) \cdot d > n\right).$$ ⌟

SPECIAL CASE: THE EQUIVALENCE REPLACEMENT RULE

Another important instance of the relation replacement rule is obtained by taking the relation $\prec$ to be the equivalence connective $\equiv$. This is possible only because we regard connectives as relations over truth values. The rule is analogous to the equality replacement rule.

Rule (equivalence replacement rule)

For any sentences $\mathcal{S}$, $\mathcal{T}$, $\tilde{\mathcal{S}}$, $\tilde{\mathcal{T}}$, $\mathcal{F}[\mathcal{S} \equiv \mathcal{T}]$, and $\mathcal{G}\langle \mathcal{S}, \mathcal{T} \rangle$, where $\mathcal{F}$ and $\mathcal{G}$ are standardized apart, we have

$$\frac{\begin{array}{c} \mathcal{F}[\mathcal{S} \equiv \mathcal{T}] \\ \mathcal{G}\langle \tilde{\mathcal{S}}, \tilde{\mathcal{T}} \rangle \end{array}}{\mathcal{F}\theta[\mathit{false}] \;\; \mathit{or} \;\; \mathcal{G}\theta\langle \mathcal{T}\theta, \mathcal{S}\theta \rangle}$$

where θ is a simultaneous, most-general unifier of $\mathcal{S}$, $\tilde{\mathcal{S}}$ and of $\mathcal{T}$, $\tilde{\mathcal{T}}$. ⌟

As in the equality replacement rule, we do not need to restrict the polarities of the replaced subsentences $\mathcal{S}\theta$ and $\mathcal{T}\theta$ in $\mathcal{G}\theta$, because any subsentence has both polarities with respect to the equivalence relation. The polarity strategy is the same as for the general relation-replacement rule.

The following illustration of the equivalence replacement rule (or $\equiv$-replacement rule) is drawn from the derivation of a program to find the maximum of a list of numbers (e.g., integers or reals).

Example

In the theory of lists of (say) integers, suppose our deduced set contains the sentences

$$\mathcal{F}: \quad \begin{array}{l} \mathit{if}\ \mathit{not}\,(x = \{\ \}) \\ \mathit{then}\ \left[\boxed{u \in x} \equiv [u = h\ \ \mathit{or}\ \ u \in t]\right]^{+} \end{array}$$

and

$$\mathcal{G}: \quad \mathit{not}\ \begin{bmatrix} z \in s\ \ \mathit{and} \\ \begin{bmatrix} \mathit{if}\ \boxed{g(z) \in s} \\ \mathit{then}\ \ z \geq g(z) \end{bmatrix} \end{bmatrix}.$$

(In the derivation, $\mathcal{F}$ is an axiom and $\mathcal{G}$ is deduced from the negation of the theorem.)

Note that the boxed subsentences $u \in x$ and $g(z) \in s$ are unifiable, with most-general unifier

$$\theta: \quad \{u \leftarrow g(z),\ x \leftarrow s\}.$$

Therefore we can apply the ≡-replacement rule, replacing the occurrence of $g(z) \in s$ in $\mathcal{G}\theta$ with

$$g(z) = h\ \ \mathit{or}\ \ g(z) \in t,$$

to deduce

$$\begin{array}{l} \begin{bmatrix} \mathit{if}\ \mathit{not}\,(s = \{\ \}) \\ \mathit{then}\ \mathit{false} \end{bmatrix} \\ \quad \mathit{or} \\ \mathit{not}\ \begin{bmatrix} z \in s\ \ \mathit{and} \\ \begin{bmatrix} \mathit{if}\ \ [g(z) = h\ \ \mathit{or}\ \ g(z) \in t] \\ \mathit{then}\ \ z \geq g(z) \end{bmatrix} \end{bmatrix}. \end{array}$$

This sentence reduces under transformation to

$$\begin{array}{l} s = \{\ \} \\ \quad \mathit{or} \\ \mathit{not}\ \begin{bmatrix} z \in s\ \ \mathit{and} \\ \begin{bmatrix} \mathit{if}\ \ [g(z) = h\ \ \mathit{or}\ \ g(z) \in t] \\ \mathit{then}\ \ z \geq g(z) \end{bmatrix} \end{bmatrix}. \end{array}$$

⌟

6. THE RELATION-MATCHING RULE

We are about to introduce not a rule in itself but an augmentation of the other rules. The resolution and relation replacement rules draw a conclusion when one subexpression in our proof unifies with another. The relation-matching augmentation allows these rules to apply even if the two expressions fail to unify, provided that certain conditions can be introduced into the conclusion. We begin by describing the augmentation of the resolution rule.

RESOLUTION WITH RELATION MATCHING: GROUND VERSION

This rule is our generalization of the E-resolution rule. The ground version of the rule is as follows:

Rule (resolution with relation matching, ground version)

For any binary relation $\prec$, ground expressions s and t, and ground sentences $\mathcal{P}\langle s^+,\ t^+,\ s^-,\ t^-\rangle$, $\mathcal{F}[\mathcal{P}\langle s^+,\ s^+,\ t^-,\ t^-\rangle]$, and $\mathcal{G}[\mathcal{P}\langle t^+,\ t^+,\ s^-,\ s^-\rangle]$ we have

$$\frac{\begin{array}{l}\mathcal{F}[\mathcal{P}\langle s^+,\ s^+,\ t^-,\ t^-\rangle]\\ \mathcal{G}[\mathcal{P}\langle t^+,\ t^+,\ s^-,\ s^-\rangle]\end{array}}{\begin{array}{l}\textit{if } s \preceq t\\ \textit{then } \mathcal{F}[\textit{false}] \textit{ or } \mathcal{G}[\textit{true}]\end{array}}$$

Here

- $\mathcal{P}\langle s^+,\ t^+,\ s^-,\ t^-\rangle$ is an arbitrary sentence, called the *intermediate* sentence, which may have positive and negative occurrences of s and t; polarity is taken with respect to $\prec$.
- The sentence $\mathcal{F}$ may have several distinct subsentences $\mathcal{P}\langle s^+,\ s^+,\ t^-,\ t^-\rangle$, each obtained from the intermediate sentence $\mathcal{P}\langle s^+,\ t^+,\ s^-,\ t^-\rangle$ by replacing certain of the positive occurrences of t with s and certain of the negative occurrences of s with t.
- Similarly, $\mathcal{G}$ may have several distinct subsentences $\mathcal{P}\langle t^+,\ t^+,\ s^-,\ s^-\rangle$, each obtained from the intermediate sentence by replacing certain of the positive occurrences of s with t and certain of the negative occurrences of t with s. ▟

For a particular relation $\prec$, we shall refer to the above as the resolution rule with $\prec$-matching.

Note that if all the subsentences $\mathcal{P}\langle s^+,\ s^+,\ t^-,\ t^-\rangle$ and $\mathcal{P}\langle t^+,\ t^+,\ s^-,\ s^-\rangle$ were identical, we could apply the original resolution rule, obtaining the conclusion $\big(\mathcal{F}[\textit{false}] \textit{ or } \mathcal{G}[\textit{true}]\big)$. The augmented rule allows us to derive the same conclusion rule even if the subsentences $\mathcal{P}$ do not match exactly, provided that the mismatches occur between terms s and t of restricted polarity and that the condition $s \preceq t$ is introduced.

The polarity strategy allows us to apply the rule only if an occurrence of one of the sentences $\mathcal{P}\langle s^+, s^+, t^-, t^-\rangle$ is positive or of no polarity in $\mathcal{F}$ and if an occurrence of one of the sentences $\mathcal{P}\langle t^+,\ t^+,\ s^-,\ s^-\rangle$ is negative or of no polarity in $\mathcal{G}$.

Note that the intermediate sentence $\mathcal{P}\langle s^+,\ t^+,\ s^-,\ t^-\rangle$ does not necessarily appear in either of the sentences of the deduced set and that the rule does not stipulate how to find such a sentence. We shall discuss the choice of the intermediate sentence in the subsection **Selection of Application Parameters.**

Example

In the theory of lists, suppose that our deduced set includes the sentences

$$\mathcal{F}: \quad p(\ell) \textit{ or } \boxed{c \in \big(\textit{tail}(\ell)\big)^+}^{\,+}$$

and

$$\mathcal{G}: \quad \textit{if } \boxed{c \in \ell^+}^{\,-} \quad \textit{then } q(\ell).$$

The two boxed subsentences are not identical. Let us take our intermediate sentence to be one of them, $\mathcal{P} : c \in \textit{tail}(\ell)$. The subterm $s^+ : \textit{tail}(\ell)$ is positive in $c \in \textit{tail}(\ell)$ with respect to the proper-sublist relation $\prec_{list}$. The other boxed subsentence $c \in \ell$ can be obtained by replacing this subterm with $t^+ : \ell$. Therefore we can apply the resolution rule with $\prec_{list}$-matching to obtain

$$\begin{array}{l}\textit{if } \textit{tail}(\ell) \preceq_{list} \ell\\ \textit{then } p(\ell) \textit{ or false}\\ \qquad\quad \textit{or}\\ \qquad \textit{if true then } q(\ell),\end{array}$$

which reduces under transformation to

$$\begin{array}{l} \textit{if } tail(\ell) \preceq_{list} \ell \\ \textit{then } p(\ell) \textit{ or } q(\ell). \end{array}$$ ∎

We shall give some more complex examples of the application of the rule after we establish its soundness.

Justification (resolution with relation matching, ground version)

Note that (by the invertibility of partial replacement) the intermediate sentence $\mathcal{P}\langle s^+, t^+, s^-, t^-\rangle$ can be obtained from any of the subsentences $\mathcal{P}\langle s^+, s^+, t^-, t^-\rangle$ of $\mathcal{F}$ by replacing certain positive occurrences of s with t and certain negative occurrences of t with s, where polarity is taken in $\mathcal{P}$ with respect to $\prec$. Therefore (by the *polarity replacement* proposition) each of the sentences

$$(\dagger) \qquad \begin{array}{l} \textit{if } s \preceq t \\ \textit{then if } \mathcal{P}\langle s^+, s^+, t^-, t^-\rangle \\ \qquad \textit{then } \mathcal{P}\langle s^+, t^+, s^-, t^-\rangle \end{array}$$

is valid.

Also any of the subsentences $\mathcal{P}\langle t^+, t^+, s^-, s^-\rangle$ of $\mathcal{G}$ can be obtained from the intermediate sentence $\mathcal{P}\langle s^+, t^+, s^-, t^-\rangle$ by replacing certain positive occurrences of s with t and certain negative occurrences of t with s. Therefore (by the *polarity replacement* proposition again) each of the sentences

$$(\ddagger) \qquad \begin{array}{l} \textit{if } s \preceq t \\ \textit{then if } \mathcal{P}\langle s^+, t^+, s^-, t^-\rangle \\ \qquad \textit{then } \mathcal{P}\langle t^+, t^+, s^-, s^-\rangle \end{array}$$

is valid.

Suppose that the sentences $\mathcal{F}[\mathcal{P}\langle s^+, s^+, t^-, t^-\rangle]$ and $\mathcal{G}[\mathcal{P}\langle t^+, t^+, s^-, s^-\rangle]$ are true and that $s \preceq t$. We would like to show that then $(\mathcal{F}[\textit{false}] \textit{ or } \mathcal{G}[\textit{true}])$ is true. The proof distinguishes between two cases, depending on whether the intermediate sentence $\mathcal{P}\langle s^+, t^+, s^-, t^-\rangle$ is false or true. We show that in each case one of the two disjuncts, $\mathcal{F}[\textit{false}]$ or $\mathcal{G}[\textit{true}]$, is true.

Case: $\mathcal{P}\langle s^+, t^+, s^-, t^-\rangle$ is false

Then by our previous conclusion ($\dagger$), because $s \preceq t$, we know each of the subsentences $\mathcal{P}\langle s^+, s^+, t^-, t^-\rangle$ of $\mathcal{F}$ is false. Because $\mathcal{F}[\mathcal{P}\langle s^+, s^+, t^-, t^-\rangle]$ is true and because the subsentences $\mathcal{P}\langle s^+, s^+, t^-, t^-\rangle$ and *false* all have the same truth value, we know (by the *value* property) that the first disjunct, $\mathcal{F}[\textit{false}]$, is true.

Case: $\mathcal{P}\langle s^+, t^+, s^-, t^-\rangle$ is true

Then by our previous conclusion ($\ddagger$), because $s \preceq t$, we know each of the sentences $\mathcal{P}\langle t^+, t^+, s^-, s^-\rangle$ is true. Because $\mathcal{G}[\mathcal{P}\langle t^+, t^+, s^-, s^-\rangle]$ is true and because $\mathcal{P}\langle t^+, t^+, s^-, s^-\rangle$ and *true* have the same truth value, we know (by the *value* property again) that the second disjunct, $\mathcal{G}[\textit{true}]$, is true. ∎

The resolution rule with relation matching must be regulated with strict heuristic controls; if the controls are too permissive, any two subsentences may be matched.

The following example is a bit contrived but illustrates some of the power of the rule.

Example

In the theory of sets, suppose our deduced set includes the two sentences

$$\mathcal{F}: \quad \begin{array}{l} \boxed{e \in \left((s^+ \sim a) \cup (b \sim t^-) \cup (t^+ \sim c) \cup (d \sim t^-)\right)}^{\,+} \\ \quad \textit{or} \\ \boxed{e \in \left((s^+ \sim a) \cup (b \sim s^-) \cup (s^+ \sim c) \cup (d \sim t^-)\right)}^{\,+} \end{array}$$

and

$$\mathcal{G}: \quad \textit{not} \left[\begin{array}{l} \boxed{e \in \left((t^+ \sim a) \cup (b \sim s^-) \cup (t^+ \sim c) \cup (d \sim t^-)\right)}^{\,-} \\ \quad \textit{and} \\ \boxed{e \in \left((s^+ \sim a) \cup (b \sim s^-) \cup (t^+ \sim c) \cup (d \sim s^-)\right)}^{\,-} \end{array}\right].$$

Let us take our intermediate sentence to be

$$\mathcal{P}: \quad e \in \left((s^+ \sim a) \cup (b \sim s^-) \cup (t^+ \sim c) \cup (d \sim t^-)\right).$$

The occurrences of s and t have been annotated with their polarities in $\mathcal{P}$ with respect to the proper-subset relation $\subset$. Note that each of the boxed sentences in $\mathcal{F}$ may be obtained from $\mathcal{P}$ by replacing certain of the positive occurrences of t with s and certain of the negative occurrences of s with t. Also, each of the boxed subsentences of $\mathcal{G}$ may be obtained from $\mathcal{P}$ by replacing certain of the positive occurrences of s with t and certain of the negative occurrences of t with s. Therefore we can apply the resolution rule with $\subset$-matching to obtain

$$\begin{array}{l} \textit{if } s \subseteq t \\ \textit{then false or false} \\ \qquad \textit{or} \\ \quad \textit{not}\,(\textit{true and true}), \end{array}$$

which reduces under transformation to the sentence

$$\textit{not}\,(s \subseteq t). \quad \blacksquare$$

Note that this conclusion, obtained by a single application of the rule, is not immediately evident to the human reader.

SPECIAL CASE: RESOLUTION WITH EQUALITY MATCHING

In the case in which the relation $\prec$ is taken to be the equality relation $=$, the resolution rule with relation matching reduces to a nonclausal variant of the E-resolution rule. It may be expressed (in the ground version) as follows:

Rule (resolution with equality matching)

For any terms s and t and sentences $\mathcal{P}\langle s, t, s, t\rangle$, $\mathcal{F}[\mathcal{P}\langle s, s, t, t, \rangle]$, and $\mathcal{G}[\mathcal{P}\langle t, t, s, s\rangle]$, we have

$$\begin{array}{c} \mathcal{F}[\mathcal{P}\langle s, s, t, t\rangle] \\ \mathcal{G}[\mathcal{P}\langle t, t, s, s\rangle] \\ \hline \begin{array}{l} \textit{if } s = t \\ \textit{then } \mathcal{F}[\textit{false}] \textit{ or } \mathcal{G}[\textit{true}]. \end{array} \end{array} \quad \blacksquare$$

Here $\mathcal{P}\langle s, s, t, t\rangle$ and $\mathcal{P}\langle t, t, s, s\rangle$ are obtained from $\mathcal{P}\langle s, t, s, t\rangle$ by replacing certain occurrences of s with t and certain occurrences of t with s. In other words, all the subsentences $\mathcal{P}\langle s, s, t, t\rangle$ and $\mathcal{P}\langle t, t, s, s\rangle$ are identical except that one may have occurrences of s where another has occurrences of t. We do not need to restrict the polarities, because every subterm of a sentence is both positive and negative with respect to the equality relation.

MULTIPLE MISMATCHED SUBSENTENCES

The resolution rule with relation matching can be extended to allow several corresponding pairs of subexpressions $s_1, t_1, s_2, t_2, \ldots$ and s_n, t_n rather than a single pair s, t, and several binary relations $\prec_1, \prec_2$ $, \ldots,$ and $\prec_n$ rather than a single binary relation $\prec$. To write the extended rule succinctly, we abbreviate $s_1, s_2, \ldots, s_n$ as $\hat{s}, t_1, t_2, \ldots, t_n$ as $\hat{t}, \prec_1, \prec_2, \ldots,$ and $\prec_n$ as $\hat{\prec}$, and

$$s_1 \preceq_1 t_1 \ \textit{and} \ s_2 \preceq_2 t_2 \ \textit{and} \ \ldots \ \textit{and} \ s_n \preceq_n t_n \quad \text{as} \quad \hat{s} \ \hat{\preceq} \ \hat{t}.$$

Then for any binary relations $\hat{\prec}$, expressions $\hat{s}$ and $\hat{t}$, and sentences $\mathcal{P}\langle \hat{s}^+, \hat{t}^+, \hat{s}^-, \hat{t}^-\rangle$, $\mathcal{F}[\mathcal{P}\langle \hat{s}^+, \hat{s}^+, \hat{t}^-, \hat{t}^-\rangle]$, and $\mathcal{G}[\mathcal{P}\langle \hat{t}^+, \hat{t}^+, \hat{s}^-, \hat{s}^-\rangle]$, we have

$$\begin{array}{l}
\mathcal{F}[\mathcal{P}\langle \hat{s}^+, \hat{s}^+, \hat{t}^-, \hat{t}^-\rangle] \\
\mathcal{G}[\mathcal{P}\langle \hat{t}^+, \hat{t}^+, \hat{s}^-, \hat{s}^-\rangle] \\
\hline
\textit{if} \ \hat{s} \ \hat{\preceq} \ \hat{t} \\
\textit{then} \ \mathcal{F}[\textit{false}] \ \textit{or} \ \mathcal{G}[\textit{true}].
\end{array}$$

The extended rule is easily justified, given the soundness of the original rule.

RESOLUTION WITH RELATION MATCHING: GENERAL VERSION

The general version of the rule allows us to instantiate the variables of the given sentences as necessary and then to apply the ground version. The precise statement, which we omit, is analogous to the precise statement of the general version of the resolution rule. We illustrate the application of the general rule with an example.

Example

Suppose our deduced set contains the sentences

$$\mathcal{F}: \quad \begin{array}{l} \textit{if} \ q(u) \\ \textit{then} \ \boxed{p(u^+, u^+)}^{+} \end{array}$$

and

$$\mathcal{G}: \quad \textit{not} \ \boxed{p(\ell^+, f(\ell)^+)}^{-}.$$

Here the annotations of the subterms within the boxed subsentences indicate their polarity in these subsentences with respect to a binary relation $\prec$.

The substitution $\theta : \{u \leftarrow \ell\}$ fails to unify the boxed subsentences of $\mathcal{F}$ and $\mathcal{G}$; the results of applying θ to these subsentences are the sentences $p(\ell^+, \ell^+)$ and $p(\ell^+, f(\ell)^+)$, respectively. Note that the mismatched occurrences of ℓ and $f(\ell)$ are positive in these sentences with respect to $\prec$.

To apply the ground version of the rule to $\mathcal{F}\theta$ and $\mathcal{G}\theta$, let us take the intermediate sentence to be $p(\ell^+, \ell^+)$. We obtain

$$\begin{array}{l} \textit{if } \ell \preceq f(\ell) \\ \textit{then } \begin{bmatrix} \textit{if } q(\ell) \\ \textit{then false} \end{bmatrix} \textit{ or } (\textit{not true}), \end{array}$$

which reduces under *true-false* transformation to

$$\begin{array}{l} \textit{if } \ell \preceq f(\ell) \\ \textit{then not } q(\ell). \quad \lrcorner \end{array}$$

SELECTION OF APPLICATION PARAMETERS

For each application of the resolution rule with relation matching, we must select the *application parameters*, i.e., the substitution θ, the intermediate sentence $\mathcal{P}$, and the subexpressions s and t. In fact, a satisfactory choice of application parameters is not straightforward: it depends on what other sentences are in the deductive set. Some considerations influencing the decision are illustrated in the next few sections.

Choice of Substitution

The substitution θ and the intermediate sentence $\mathcal{P}$ for applying the rule are not necessarily unique.

In the example above, consider again the boxed subsentences $p(u^+, u^+)$ and $p(\ell^+, f(\ell)^+)$ of $\mathcal{F}$ and $\mathcal{G}$. Instead of the substitution $\theta : \{u \leftarrow \ell\}$, consider the substitution $\theta' : \{u \leftarrow f(\ell)\}$. This substitution also fails to unify the boxed subsentences; the results of applying θ' to the boxed subsentences are the sentences $p(f(\ell)^+, f(\ell)^+)$ and $p(\ell^+, f(\ell)^+)$, respectively. Note that the mismatched occurrences of $f(\ell)$ and ℓ are positive in these sentences with respect to $\prec$.

To apply the ground version of the rule to $\mathcal{F}\theta'$ and $\mathcal{G}\theta'$, let us take the intermediate sentence to be $p(f(\ell)^+,\ f(\ell)^+)$. We obtain

$$\begin{array}{l} \textit{if } f(\ell) \preceq \ell \\ \textit{then } \begin{bmatrix} \textit{if } q(\ell) \\ \textit{then false} \end{bmatrix} \textit{ or } (\textit{not true}), \end{array}$$

which reduces under *true-false* transformation to

$$\begin{array}{l} \textit{if } f(\ell) \preceq \ell \\ \textit{then not } q(\ell). \end{array}$$

This is not equivalent to the sentence we obtained by applying the rule with the substitution θ,

$$\begin{array}{l} \textit{if } \ell \preceq f(\ell) \\ \textit{then not } q(\ell). \end{array}$$

In other words, we must consider both ways of applying the rule.

To Unify or Not to Unify

In previous examples, we have applied the resolution rule with relation matching only when it is illegal to apply the ordinary resolution rule because the matched subsentences fail to unify. In some cases, however, we must use relation matching to obtain a refutation even though the matched subsentences do unify and the resolution rule could be applied.

For example, suppose our deduced set consists of the sentences

1. $\boxed{p(x^+)}$ *or* $q(x^+)$

2. *not* $\boxed{p(a^+)}^-$

3. *not* $\boxed{q(b^+)}^-$

4. $c \preceq a$

5. $c \preceq b$,

where x is positive in the boxed subsentence $p(x)$ and in the subsentence $q(x)$ with respect to the relation $\prec$, as indicated by its annotation.

It is legal to apply the ordinary resolution rule to the first two sentences, taking the unifier to be $\{x \leftarrow a\}$, to deduce (after transformation)

$$q(a).$$

However, this sentence is of no use in a refutation.

If instead we apply the resolution rule with $\prec$-matching to the same boxed subsentences, taking the unifier to be the empty substitution $\{\ \}$, we obtain (after transformation)

6. *if* $x \preceq a$ *then* $\boxed{q(x^+)}^+$.

We can then apply the resolution rule to sentences 6 and 3, taking the unifier to be the empty substitution $\{\ \}$, to obtain (after transformation)

7. *if* $x \preceq b$ *then* *not* $(x \preceq a)$.

We finally obtain a refutation by applying the resolution rule to this sentence and the last two sentences in turn; the unifier is $\{x \leftarrow c\}$.

In applying the ordinary resolution rule, we committed x to be a; this turned out to be a mistake. In applying the resolution rule with $\prec$-matching instead, we left x free to be any element such that $x \preceq a$; in particular, we could then take x to be c.

Choice of Mismatched Subexpressions

In the examples of resolution with relation matching we have seen, we have always taken the mismatched subexpressions s and t to be as small as possible. Sometimes this choice costs us a proof.

For instance, suppose our deduced set consists of the sentences

1. $\boxed{p(f(a))}^+$

2. *not* $\boxed{p(f(b))}^-$

3. $f(a) = f(b)$.

If we apply the resolution rule with equality matching to the first two sentences, taking s to be a and t to be b, we obtain

if $a = b$
then *false* *or* *not* *true*,

which reduces under transformation to

not $(a = b)$.

This sentence is of no use in a refutation.

On the other hand, if instead we apply the same rule taking s to be $f(a)$ and t to be $f(b)$, we obtain

$$\begin{array}{l} \textit{if } f(a) = f(b) \\ \textit{then false or not true}, \end{array}$$

which reduces under transformation to

$$\textit{not}\,\big(f(a) = f(b)\big).$$

A refutation can be obtained immediately by applying the resolution rule to the third sentence and this one.

In the preceding examples, we have seen that in applying the resolution rule with relation matching, the choice of appropriate application parameters, i.e., the substitution θ, the intermediate sentence $\mathcal{P}$, and the mismatched subexpressions s and t, are not unique and depend on the other sentences in the deduced set. Digricoli [83] provides an algorithm to generate all legal sets of application parameters. This algorithm is phrased in terms of his variant of the E-resolution rule but extends readily to the general, nonclausal case. Digricoli also suggests a heuristic *viability criterion* for selecting a single appropriate set of application parameters; this criterion appears to extend to the general case as well.

REPLACEMENT WITH RELATION MATCHING: GROUND VERSION

We have shown how to augment the resolution rule to apply even if the matched subsentences are not entirely unified by the substitution. We now introduce an analogous augmentation of the relation replacement rule.

Rule (replacement with relation matching, ground version)

For any binary relations $\prec_1$ and $\prec_2$, ground expressions s, t, $u\langle s^+, t^+, s^-, t^-\rangle$, and $v\langle s^+, t^+, s^-, t^-\rangle$, and ground sentences

$$\mathcal{F}[u\langle s^+,\ s^+,\ t^-,\ t^-\rangle \prec_1 v\langle s^+,\ s^+,\ t^-,\ t^-\rangle]$$

and

$$\mathcal{G}\langle u\langle t^+,\ t^+,\ s^-,\ s^-\rangle^+,\ v\langle t^+,\ t^+,\ s^-,\ s^-\rangle^-\rangle,$$

we have

$$\begin{array}{l} \mathcal{F}[u\langle s^+,\ s^+,\ t^-,\ t^-\rangle \prec_1 v\langle s^+,\ s^+,\ t^-,\ t^-\rangle] \\ \mathcal{G}\langle u\langle t^+,\ t^+,\ s^-,\ s^-\rangle^+,\ \ v\langle t^+,\ t^+,\ s^-,\ s^-\rangle^-\rangle \\ \hline \textit{if } s \preceq_2 t \\ \textit{then } \mathcal{F}[\textit{false}] \textit{ or } \mathcal{G}\langle v\langle t^+,\ t^+,\ s^-,\ s^-\rangle^+,\ \ u\langle t^+,\ t^+,\ s^-,\ s^-\rangle^-\rangle \end{array}$$

Here

- The expressions $u\langle s^+,\ t^+,\ s^-,\ t^-\rangle$ and $v\langle s^+,\ t^+,\ s^-,\ t^-\rangle$ are arbitrary expressions. The sentence $u\langle s^+,\ t^+,\ s^-,\ t^-\rangle \prec_1 v\langle s^+,\ t^+,\ s^-,\ t^-\rangle$ is called the *intermediate sentence*.

- The subsentences $u\langle s^+,\ s^+,\ t^-,\ t^-\rangle \prec_1 v\langle s^+,\ s^+,\ t^-,\ t^-\rangle$ of $\mathcal{F}$ are obtained from the intermediate sentence by replacing certain positive occurrences of t with s and certain negative occurrences of s with t, where polarity is taken in the intermediate sentence with respect to $\prec_2$.

- The subexpressions $u\langle t^+,\ t^+,\ s^-,\ s^-\rangle$ and $v\langle t^+,\ t^+,\ s^-,\ s^-\rangle$ of $\mathcal{G}$ are obtained from $u\langle s^+, t^+, s^-, t^-\rangle$ and $v\langle s^+, t^+, s^-, t^-\rangle$, respectively, by replacing certain occurrences of s

with t and certain occurrences of t with s, where again polarity is taken in the intermediate sentence with respect to $\prec_2$.

- The subsentence $\mathcal{G}\langle v\langle t^+, t^+, s^-, s^-\rangle^+, u\langle t^+, t^+, s^-, s^-\rangle^-\rangle$ of the conclusion is obtained from $\mathcal{G}\langle u\langle t^+, t^+, s^-, s^-\rangle^+, v\langle t^+, t^+, s^-, s^-\rangle^-\rangle$ by replacing certain positive occurrences of $u\langle t^+, t^+, s^-, s^-\rangle$ with $v\langle t^+, t^+, s^-, s^-\rangle$ and certain negative occurrences of $v\langle t^+, t^+, s^-, s^-\rangle$ with $u\langle t^+, t^+, s^-, s^-\rangle$, where the polarity of u and v is taken in $\mathcal{G}$ with respect to $\prec_1$. $\lrcorner$

For particular binary relations $\prec_1$ and $\prec_2$, we shall call this the *$\prec_1$-replacement rule with $\prec_2$-matching.* Note that if $u\langle t^+, t^+, s^-, s^-\rangle$ and $v\langle t^+, t^+, s^-, s^-\rangle$ were identical to $u\langle s^+, s^+, t^-, t^-\rangle$ and $v\langle s^+, s^+, t^-, t^-\rangle$, respectively, we could apply the original $\prec_1$-replacement rule without $\prec_2$-matching, obtaining the conclusion

$$\mathcal{F}[false] \;\; or \;\; \mathcal{G}\langle v\langle t^+, t^+, s^-, s^-\rangle^+, u\langle t^+, t^+, s^-, s^-\rangle^-\rangle.$$

The augmented rule allows us to derive the same conclusion, even if the subexpressions do not match exactly, provided that the mismatches occur between subexpressions s and t of restricted polarity with respect to $\prec_2$ and that the condition $s \preceq_2 t$ is added.

Example

In a theory that includes the lists and the integers, suppose our deduced set contains the sentences

$$\mathcal{F}: \quad \left(\boxed{length(m^-)}^- \le a\right) \;\; or \;\; p(m)$$

and

$$\mathcal{G}: \quad if \;\; q(\ell) \;\; then \;\; \left(\boxed{length(\ell^-)}^+ > b\right),$$

where ℓ and m are lists and a and b are integers.

The two boxed subexpressions are not identical, so we cannot apply the original $\le$-replacement rule. To apply the augmented rule, let us take our intermediate sentence to be $length(\ell) \le a$. With respect to the proper sublist relation $\prec_{list}$, the subterm $s^- : \ell$ is negative in the intermediate sentence $u \prec_1 v : length(\ell) \le a$. From this sentence we can obtain the subsentence $length(m) \le a$ of $\mathcal{F}$ by replacing the negative occurrence of ℓ with $t^- : m$. Therefore, by the $\le$-replacement rule with $\prec_{list}$-matching, we deduce

$$\begin{array}{l} if \;\; \ell \preceq_{list} m \\ then \;\; false \;\; or \;\; p(m) \\ \qquad\quad or \\ \qquad if \;\; q(\ell) \;\; then \;\; a > b. \end{array}$$

Here the subsentence $a > b$ of the conclusion is obtained from the subsentence $length(\ell) > b$ of $\mathcal{G}$ by replacing a positive occurrence of $u^+ : length(\ell)$ with $v^+ : b$, where polarity is taken in $\mathcal{G}$ with respect to the weak less-than relation $\le$. The conclusion reduces under transformation to

$$\begin{array}{l} if \;\; \ell \preceq_{list} m \\ then \;\; p(m) \;\; or \\ \qquad if \;\; q(\ell) \;\; then \;\; a > b. \end{array} \quad \lrcorner$$

Now let us establish the soundness of the rule.

Justification (replacement with relation matching, ground version)

Note that (by the invertibility of partial replacements), the intermediate sentence $u\langle s^+, t^+, s^-, t^-\rangle \prec_1 v\langle s^+, t^+, s^-, t^-\rangle$ can be obtained from any of the subsentences $u\langle s^+, s^+, t^-, t^-\rangle \prec_1 v\langle s^+, s^+, t^-, t^-\rangle$ of

$\mathcal{F}$ by replacing certain positive occurrences of s with t and certain negative occurrences of t with s, where polarity is taken in the subsentences with respect to $\prec_2$. Therefore (by the *polarity replacement* proposition), each of the sentences

$$(\dagger)\qquad \begin{array}{l} \textit{if } s \preceq_2 t \\ \textit{then if } u\langle s^+,\ s^+,\ t^-,\ t^-\rangle \prec_1 v\langle s^+,\ s^+,\ t^-,\ t^-\rangle \\ \qquad \textit{then } u\langle s^+,\ t^+,\ s^-,\ t^-\rangle \prec_1 v\langle s^+,\ t^+,\ s^-,\ t^-\rangle \end{array}$$

is valid.

Also any of the sentences $u\langle t^+, t^+, s^-, s^-\rangle \prec_1 v\langle t^+, t^+, s^-, s^-\rangle$ can be obtained from the intermediate sentence $u\langle s^+, t^+, s^-, t^-\rangle \prec_1 v\langle s^+, t^+, s^-, t^-\rangle$ by replacing certain positive occurrences of s with t and certain negative occurrences of t with s, where polarity is taken in the intermediate sentence with respect to $\prec_2$. Therefore (by the *polarity replacement* proposition again) each of the sentences

$$(\ddagger)\qquad \begin{array}{l} \textit{if } s \preceq_2 t \\ \textit{then if } u\langle s^+,\ t^+,\ s^-,\ t^-\rangle \prec_1 v\langle s^+,\ t^+,\ s^-,\ t^-\rangle \\ \qquad \textit{then } u\langle t^+,\ t^+,\ s^-,\ s^-\rangle \prec_1 v\langle t^+,\ t^+,\ s^-,\ s^-\rangle \end{array}$$

is valid.

Furthermore the subsentence $\mathcal{G}\langle v\langle t^+, t^+, s^-, s^-\rangle^+, u\langle t^+, t^+, s^-, s^-\rangle^-\rangle$ of the conclusion can be obtained from the given sentence $\mathcal{G}\langle u\langle t^+, t^+, s^-, s^-\rangle^+, v\langle t^+, t^+, s^-, s^-\rangle^-\rangle$ of the deduced set by replacing certain positive occurrences of $u\langle t^+, t^+, s^-, s^-\rangle$ with $v\langle t^+, t^+, s^-, s^-\rangle$ and certain negative occurrences of $v\langle t^+, t^+, s^-, s^-\rangle$ with $u\langle t^+, t^+, s^-, s^-\rangle$, where polarity is taken in $\mathcal{G}$ with respect to $\prec_1$. Therefore (by the *polarity replacement* proposition once again) each of the sentences

$$(\dagger\dagger)\qquad \begin{array}{l} \textit{if } u\langle t^+,\ t^+,\ s^-,\ s^-\rangle \prec_1 v\langle t^+,\ t^+,\ s^-,\ s^-\rangle \\ \textit{then if } \mathcal{G}\langle u\langle t^+,\ t^+,\ s^-,\ s^-\rangle^+,\ v\langle t^+,\ t^+,\ s^-,\ s^-\rangle\rangle \\ \qquad \textit{then } \mathcal{G}\langle v\langle t^+,\ t^+,\ s^-,\ s^-\rangle^+,\ u\langle t^+,\ t^+,\ s^-,\ s^-\rangle^-\rangle \end{array}$$

is valid.

Suppose that the ground sentences

$$\mathcal{F}[u\langle s^+,\ s^+,\ t^-,\ t^-\rangle \prec_1 v\langle s^+,\ s^+,\ t^-,\ t^-\rangle] \quad \text{and} \quad \mathcal{G}\langle u\langle t^+,\ t^+,\ s^-,\ s^-\rangle^+,\ v\langle t^+,\ t^+,\ s^-,\ s^-\rangle^-\rangle$$

are true and that $s \preceq_2 t$. We would like to show that then

$$\mathcal{F}[\textit{false}] \ \textit{ or } \ \mathcal{G}\langle v\langle t^+,\ t^+,\ s^-,\ s^-\rangle^+,\ u\langle t^+,\ t^+,\ s^-,\ s^-\rangle^-\rangle$$

is true. The proof distinguishes between two cases, depending on whether the intermediate sentence is false or true. We show that in each case one of the two disjuncts, $\mathcal{F}[\textit{false}]$ or $\mathcal{G}\langle v\langle t^+, t^+, s^-, s^-\rangle^+, u\langle t^+, t^+, s^-, s^-\rangle^-\rangle$, is true.

Case: $u\langle s^+, t^+, s^-, t^-\rangle \prec_1 v\langle s^+, t^+, s^-, t^-\rangle$ is false

Then by our previous conclusion (†), because $s \preceq_2 t$, we know each of the subsentences $u\langle s^+, s^+, t^-, t^-\rangle \prec_1 v\langle s^+, s^+, t^-, t^-\rangle$ of $\mathcal{F}$ is false. Because $\mathcal{F}[u\langle s^+, s^+, t^-, t^-\rangle \prec_1 v\langle s^+, s^+, t^-, t^-\rangle]$ is true and because the sentences $u\langle s^+, s^+, t^-, t^-\rangle \prec_1 v\langle s^+, s^+, t^-, t^-\rangle$ and *false* all have the same truth value, we know (by the *value* property) that the first disjunct, $\mathcal{F}[\textit{false}]$, is true.

Case: $u\langle s^+, t^+, s^-, t^-\rangle \prec_1 v\langle s^+, t^+, s^-, t^-\rangle$ is true

Then by our previous conclusion (‡), because $s \preceq_2 t$, we know each of the sentences $u\langle t^+, t^+, s^-, s^-\rangle \prec_1 v\langle t^+, t^+, s^-, s^-\rangle$ is true. Therefore by several applications of our previous conclusion (††), because

$$\mathcal{G}\langle u\langle t^+,\ t^+,\ s^-,\ s^-\rangle^+,\ v\langle t^+,\ t^+,\ s^-,\ s^-\rangle^-\rangle$$

is true, we know that the second disjunct,

$$\mathcal{G}\langle v\langle t^+,\ t^+,\ s^-,\ s^-\rangle^+,\ u\langle t^+,\ t^+,\ s^-,\ s^-\rangle^-\rangle,$$

is true.

In each case, we have shown that the desired conclusion is true. ┛

REPLACEMENT WITH RELATION MATCHING: GENERAL VERSION

The general version of the rule allows us to instantiate the variables of the given sentences as necessary and then to apply the ground version. We omit the precise statement, which is analogous to the general version of the relation replacement rule, but we illustrate the general version with an example extracted from the derivation of a program to sort a list of numbers.

Example

In a theory of lists of (say) integers, suppose our deduced set contains the sentences

$$\mathcal{F}:\quad \left[\boxed{perm\big(x_1 \square (\langle u\rangle \square x_2),\ y_1 \square (\langle u\rangle \square y_2)\big)} \equiv perm(x_1 \square x_2,\ y_1 \square y_2)\right]^+$$

and

$$\mathcal{G}:\quad not\,\big(ordered(z)\ \ and\ \boxed{perm(\ell^+,\ z)}\big).$$

Here the term $x_1 \square x_2$ is the result of appending the lists x_1 and x_2, and the term $\langle u\rangle$ is the list whose sole element is u. Also, $perm(\ell, z)$ holds if the list ℓ is a permutation of the list z, and $ordered(z)$ holds if the elements of z are in (weakly) increasing order. In the derivation, $\mathcal{F}$ is one of the axioms for the permutation relation, which states that two lists are permutations if they are still permutations after dropping a common element, and $\mathcal{G}$ is the negation of the theorem, which states the existence of an ordered list that is a permutation of a given list.

The results of applying the substitution

$$\theta:\quad \{z \leftarrow y_1 \square (\langle u\rangle \square y_2)\}$$

to the boxed subsentences are

$$perm\big((x_1 \square (\langle u\rangle \square x_2))^+,\ \ y_1 \square (\langle u\ \rangle \square y_2)\big)$$

and

$$perm\big(\ell^+,\ \ y_1 \square (\langle u\rangle \square y_2)\big).$$

The mismatched subterms

$$x_1 \square (\langle u\rangle \square x_2) \quad \text{and} \quad \ell$$

are positive in their respective subsentences with respect to the *perm* relation. (Because this relation is symmetric, they also happen to be negative.) The boxed subsentence $perm(\ell, z)$ is positive in $\mathcal{G}$ with respect to the equivalence relation $\equiv$. (It also happens to be negative.) Therefore, by the $\equiv$-replacement rule with *perm*-matching, we may deduce the sentence

$$\begin{array}{l} if\ perm\big(x_1 \square (\langle u\rangle \square x_2),\ \ell\big) \\ then\ false \\ \qquad\quad or \\ \qquad not\,\big(ordered(y_1 \square (\langle u\rangle \square y_2))\ \ and\ \ perm(x_1 \square x_2,\ y_1 \square y_2)\big) \end{array}$$

which reduces under transformation to

$$\begin{array}{l} \textit{if } \mathit{perm}\big(x_1 \sqcap (\langle u \rangle \sqcap x_2),\ \ell\big) \\ \textit{then } \ \mathit{not}\ \big(\mathit{ordered}(y_1 \sqcap (\langle u \rangle \sqcap y_2))\ \ \textit{and}\ \ \mathit{perm}(x_1 \sqcap x_2,\ y_1 \sqcap y_2)\big). \end{array} \quad \lrcorner$$

RELATION MATCHING VERSUS RELATION REPLACEMENT

The relation matching and relation replacement rules play complementary roles, and one might expect that a single deductive system would employ one or the other rule but not both. After all, in clausal equality systems, paramodulation and a variant of E-resolution have each been shown to be complete (Anderson [70], Digricoli [83], and Brand [75]) without including the other. Moreover, by incorporating both rules, we admit a troublesome redundancy: The same conclusion can be derived in several ways.

On the other hand, it often turns out that a proof that seems unmotivated or tricky using only one of the rules seems more straightforward using a combination of both. For instance, in an example of a previous section, we applied the resolution rule with relation matching to the sentences

$$\mathcal{F}: \quad \begin{array}{l} \textit{if } q(u) \\ \textit{then } \boxed{p(u^+,\ u^+)}^{+} \end{array}$$

and

$$\mathcal{G}: \quad \mathit{not}\,\boxed{p(\ell^+, f(\ell)^+)}^{-}$$

taking the substitution to be

$$\theta: \quad \{u \leftarrow \ell\},$$

to obtain after transformation

$$\begin{array}{l} \textit{if } \boxed{\ell \prec f(\ell)} \\ \textit{then } \ \mathit{not}\, q(\ell). \end{array}$$

If our deduced set also contains the sentence

$$\boxed{v \prec f(v)},$$

we can further deduce (by resolution) the sentence

$$\mathit{not}\, q(\ell).$$

Now suppose our deductive system includes the relation replacement rule but not the relation-matching rule. Then to deduce the same conclusion *not* $q(\ell)$, we would have to apply the relation replacement rule to the sentences

$$\boxed{v}^{+} \prec f(v)$$

and

$$\mathcal{G}: \quad \mathit{not}\, p(\boxed{\ell^+},\ f(\ell))$$

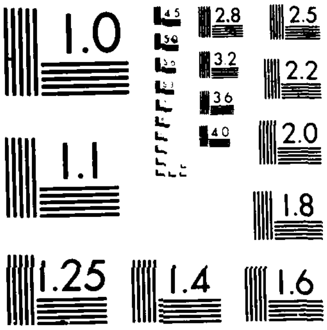
to obtain (after transformation)

$$\mathit{not}\,\boxed{p(f(\ell),\ f(\ell))}$$

AD-A175 249 A DEDUCTIVE APPROACH TO COMPUTER PROGRAMMING(U) 2/2
STANFORD UNIV CA DEPT OF COMPUTER SCIENCE Z MANNA 1986
AFOSR-TR-86-2164 AFOSR-81-0014

UNCLASSIFIED F/G 9/2 NL

2 87



ROCOPY RESOLUTION TEST CHART

We could then obtain the same conclusion $(not\ q(\ell))$ by resolution applied to this sentence and the sentence

$$\mathcal{F}:\quad \begin{array}{l} if\ q(u) \\ then\ \boxed{p(u,u)}. \end{array}$$

Although both sequences of inference lead to the same conclusion, the earlier proof seems better motivated: Each step is based on matching subexpressions that already possess a high degree of syntactic similarity. In contrast, the above proof seems rather gratuitous: The application of the relation replacement rule is based on matching the variable v with the constant ℓ. There is no reason to perform this step except as a preparation for the subsequent resolution step.

Examples can also be exhibited for which a proof employing the replacement rule is well-motivated but the corresponding proof using the matching rule appears strained. For instance, in the theory of integers, use of the =-replacement rule and the axiom $u + (-u) = 0$ allows us to simplify a subterm of form $t + (-t)$ to 0. If we are only permitted to use the relation-matching rule, we must leave the subterm intact, and hope that we attempt to match it against a corresponding subterm 0 later in the proof.

We expect that by including both rules together in a system we shall be able to apply more restrictive strategies to each of them. Consequently, we shall obtain a smaller search space than if we had included either of the rules separately.

7. STRENGTHENING

The relation replacement rule of Section 5 does not always allow us to draw the strongest possible conclusion. In this section we establish a stronger form of the polarity replacement lemma and use it to develop a stronger relation-replacement rule.

We motivate the strengthening of the rule with an example. In the theory of the integers, suppose our deduced set contains the sentences

$$\mathcal{F}:\quad \boxed{s} < t$$

and

$$\mathcal{G}:\quad a \leq \boxed{s}^{+} + 2.$$

Because the occurrence of s in $\mathcal{G}$ is positive with respect to the less-than relation $<$, the$<$-replacement rule allows us to replace s with t and deduce that (after transformation)

$$a \leq t^{+} + 2.$$

From these two sentences, however, we should be able to deduce the stronger result

$$a < t + 2.$$

Similarly, from the sentence $s < t$ and $not\ (a - s > b)$, we should be able to deduce $not\ (a - t \geq b)$ rather than merely $not\ (a - t > b)$.

Unfortunately, the rule as we have presented it does not yield these more useful conclusions; the strengthened relation-replacement rule will. But first, we must introduce some preliminary notions.

THE STRENGTHENED POLARITY-REPLACEMENT LEMMA

The strengthened rule depends on the following basic result:

Lemma (strengthened polarity replacement)

Consider arbitrary expressions $e\langle x, y\rangle$ and $e'\langle x, y\rangle$ and binary relations $\preceq_1$ and $\preceq_2$. The sentence

$$\begin{array}{l} \textit{if } x \preceq_1 y \\ \textit{then if } e\langle x, y\rangle \preceq_2 e'\langle x, y\rangle \\ \qquad \textit{then } e\langle y, x\rangle \prec_2 e'\langle y, x\rangle \end{array}$$

is valid provided that *the replaced occurrences of* x *and* y *satisfy the following strengthening conditions* [*in* $e\langle x, y\rangle$ *and* $e'\langle x, y\rangle$ *with respect to* $\preceq_1$ *and* $\preceq_2$]:

- *transitivity condition*

  The relation $\prec_2$, the irreflexive restriction of $\preceq_2$, is transitive.

- *top condition*

  The replaced occurrences of x and y are respectively positive and negative in $e\langle x, y\rangle \preceq_2 e'\langle x, y\rangle$ with respect to $\preceq_1$.

- *left-right condition*

  One of the following two disjuncts holds:

  The replaced occurrences of x and y in $e\langle x, y\rangle$ are respectively negative and positive in $e\langle x, y\rangle$ with respect to $\preceq_1$ and $\prec_2$ (and some replacement is made in $e\langle x, y\rangle$)

  (*left disjunct*)

  or

  the replaced occurrences of x and y in $e'\langle x, y\rangle$ are respectively positive and negative in $e'\langle x, y\rangle$ with respect to $\preceq_1$ and $\prec_2$ (and some replacement is made in $e'\langle x, y\rangle$).

  (*right disjunct*) ▟

Before proving this proposition, let us illustrate it with an example.

Example (strengthened polarity-replacement lemma)

In a theory that includes the sets and the nonnegative integers, take $\preceq_1$ to be the proper-subset relation $\subset$ over the sets and $\preceq_2$ to be the weak less-than relation $\leq$ over the nonnegative integers. Then $\prec_2$ is the strict less-than relation $<$.

Consider the sentence

$$m \cdot card(y) \leq n + card(x),$$

where x and y are sets, m and n are nonnegative integers, and $card(x)$ is the cardinality of the set x. According to the lemma, the sentence

$$\begin{array}{l} \textit{if } x \subset y \\ \textit{then if } m \cdot card(y) \leq n + card(x) \\ \qquad \textit{then } m \cdot card(x) < n + card(y) \end{array}$$

is valid, because the replaced occurrences of x and y satisfy the strengthening conditions in $m \cdot card(y)$ and $n + card(x)$ with respect to $\subset$ and $\leq$. In particular,

- The relation $<$ is transitive; hence the *transitivity* condition is satisfied.

- The replaced occurrences of x and y are respectively positive and negative in $m \cdot card(y) \leq n + card(x)$ with respect to $\subset$; hence the *top* condition is satisfied.
- Although the replaced occurrence of y is not positive in $m \cdot card(y)$ with respect to $\subset$ and $<$ (after all, m could be 0), the replaced occurrence of x is positive in $n + card(x)$ with respect to $\subset$ and $<$. Hence, though the *left* disjunct of the *left-right* condition is not satisfied, the *right* disjunct is. ⌟

We are now ready to establish the lemma.

Proof (strengthened polarity-replacement lemma)

Suppose that

$$x \preceq_1 y \quad \text{and} \quad e\langle x, y\rangle \preceq_2 e'\langle x, y\rangle,$$

and that the strengthening conditions are satisfied.

We would like to show that then

$$e\langle y, x\rangle \prec_2 e'\langle y, x\rangle.$$

The *left-right* condition was stated as a disjunction of two possibilities; we treat each possibility separately.

Case (*left disjunct*): The replaced occurrences of x and y in $e\langle x, y\rangle$ are respectively negative and positive in $e\langle x, y\rangle$ with respect to $\preceq_1$ and $\prec_2$ (and some replacement is made in $e\langle x, y\rangle$).

In this case (by the *transitive polarity-replacement* lemma, because $x \preceq_1 y$), we have

$$e\langle y, x\rangle \prec_2 e\langle x, y\rangle.$$

Also (by the *polarity replacement* proposition and our supposition that $x \preceq_1 y$ and $e\langle x, y\rangle \preceq_2 e'\langle x, y\rangle$) we have

$$e\langle x, y\rangle \preceq_2 e'\langle y, x\rangle.$$

(Here we have only performed the replacements on the right-hand side; by the *top* condition, we know the replaced occurrences of x and y are respectively positive and negative in $e\langle x, y\rangle \preceq_2 e'\langle x, y\rangle$ with respect to $\preceq_1$.) It follows that

$$e\langle x, y\rangle \prec_2 e'\langle y, x\rangle \quad \text{or} \quad e\langle x, y\rangle = e'\langle y, x\rangle.$$

Because $e\langle y, x\rangle \prec_2 e\langle x, y\rangle$, we thus have (either by the transitivity of $\prec_2$ or the substitutivity of equality) that

$$e\langle y, x\rangle \prec_2 e'\langle y, x\rangle,$$

as we wanted to show.

Case (*right disjunct*): The replaced occurrences of x and y in $e'\langle x, y\rangle$ are respectively positive and negative in $e'\langle x, y\rangle$ with respect to $\preceq_1$ and $\prec_2$ (and some replacement is made in $e\langle x, y\rangle$).

The proof in this case is entirely symmetric to the proof in the previous case. ⌟

THE STRENGTHENED POLARITY-REPLACEMENT PROPOSITION

The strengthened rule is expressed in terms of the following notational device:

Definition (strengthen accordingly)

Suppose $\prec$ is a binary relation, s and t are expressions (either both sentences or both terms), and $\mathcal{G}$ is a sentence.

If we write $\mathcal{G}$ as $\mathcal{G}\langle s^+,\ t^-\rangle$, then $\mathcal{G}\langle t^+,\ s^-\rangle^\dagger$ denotes the sentence obtained by replacing certain positive occurrences of s with t, replacing certain negative occurrences of t with s (where polarity is taken with respect to $\prec$), and *strengthening accordingly* as follows:

- Whenever a replacement is made in a positive subsentence of form $e\langle s,t\rangle \tilde{\prec}\, e'\langle s,t\rangle$, where the replaced occurrences of s and t satisfy the strengthening conditions in $e\langle s,t\rangle$ and $e'\langle s,t\rangle$ with respect to $\prec$ and $\tilde{\prec}$, replace the occurrence of the symbol $\tilde{\prec}$ with $\tilde{\prec}$, the irreflexive restriction of $\tilde{\prec}$.

- Whenever a replacement is made in a negative subsentence of form $e\langle s,t\rangle \tilde{\prec}\, e'\langle s,t\rangle$, where the replaced occurrences of s and t satisfy the strengthening conditions in $e\langle s,t\rangle$ and $e'\langle s,t\rangle$ with respect to $\prec$ and $\tilde{\not\prec}$, replace the occurrence of the symbol $\tilde{\prec}$ with $\tilde{\preceq}$. (Here $\tilde{\not\prec}$ and $\tilde{\preceq}$ are the negation, and the reflexive closure, respectively, of $\tilde{\prec}$.) ⌟

These conditions may appear mysterious at this point, but they are precisely what we need to establish the following result, which tightens up the polarity replacement proposition:

Proposition (strengthened polarity replacement)

For any binary relation $\prec$ and sentence $\mathcal{P}\langle x^+,\ y^-\rangle$, the sentence

```
if $x \prec y$
then if $\mathcal{P}\langle x^+,\ y^-\rangle$
     then $\mathcal{P}\langle y^+,\ x^-\rangle^\dagger$
```

is valid. ⌟

We illustrate the proposition with two examples.

Example

In the theory of the positive integers (excluding 0), take $\prec$ to be the proper-divides relation $\prec_{div}$ and take our sentence to be

$$\mathcal{P}\langle x^+,\ y^-\rangle: \quad a \le (x+1)^2 \ \textit{ or } \ q(x).$$

Then according to the proposition, the sentence

```
if $x \prec_{div} y$
then if $a \le (x+1)^2$ or $q(x)$
     then $a < (y+1)^2$ or $q(x)$
```

is valid. Note that the symbol $\le$ has been replaced by its irreflexive restriction $<$ as a result of the strengthening. This is because

- The subsentence $a \le (x+1)^2$ is positive in $\mathcal{P}\langle x^+,\ y^-\rangle$.
- The replaced occurrence of x in $a \le (x+1)^2$ satisfies the strengthening conditions in a and $(x+1)^2$ with respect to $\prec_{div}$ and $\le$. In particular
  - The relation $<$ is transitive; hence the *transitivity* condition is satisfied.

- The replaced occurrence of x is positive in $a \leq (x+1)^2$ with respect to $\prec_{div}$; hence the *top* condition is satisfied.
- The replaced occurrence of x is positive in $(x+1)^2$ with respect to $\prec_{div}$ and $<$; hence the *right* disjunct of the *left-right* condition is satisfied. ⌟

Example

In a theory that includes the lists and the nonnegative integers, take $\prec$ to be the tail relation $\prec_{tail}$ over the lists and take our sentence to be

$$\mathcal{P}\langle x^+,\ y^-\rangle : \quad \textit{if } length(x \Box \ell) < length(y) + m \textit{ then } q(x, y),$$

where x, y, and ℓ are lists, m is a nonnegative integer, and $length(\ell)$ is the number of elements in the list ℓ. Then according to the proposition, the sentence

$$\begin{array}{l} \textit{if } x \prec_{tail} y \\ \textit{then if if } length(x \Box \ell) < length(y) + m \textit{ then } q(x, y) \\ \qquad \textit{then if } length(y \Box \ell) \leq length(x) + m \textit{ then } q(x, y) \end{array}$$

is valid. Note that here the symbol $<$ has been replaced by $\leq$ as a result of the strengthening. This is because

- The subsentence $length(x \Box \ell) < length(y) + m$ is negative in $\mathcal{P}\langle x^+,\ y^-\rangle$.
- The replaced occurrences of x and y satisfy the strengthening conditions in $length(x \Box \ell)$ and $length(y) + m$ with respect to $\prec_{tail}$ and $\not<$, that is $\geq$. In particular
  - The relation $>$, the irreflexive restriction of $\geq$, is transitive; hence the *transitivity* condition is satisfied.
  - The replaced occurrences of x and y are positive and negative, respectively, in the sentence $length(x \Box \ell) \geq length(y) + m$ with respect to $\prec_{tail}$; hence the *top* condition is satisfied.
  - The replaced occurrence of x is negative in $length(x \Box \ell)$ with respect to $\prec_{tail}$ and $>$; hence the *left* disjunct of the *left-right* condition is satisfied. (As it turns out, the replaced occurrence of y is also negative in $length(y) + m$ with respect to $\prec_{tail}$ and $>$; hence the *right* disjunct is also satisfied.) ⌟

Let us now prove the proposition.

Proof (strengthened polarity-replacement proposition)

We suppose that

$$x \prec y \quad \text{and} \quad \mathcal{P}\langle x^+,\ y^-\rangle,$$

and show that then

$$\mathcal{P}\langle y^+,\ x^-\rangle^\dagger.$$

The sentence $\mathcal{P}\langle y^+,\ x^-\rangle^\dagger$ is obtained from $\mathcal{P}\langle x^+,\ y^-\rangle$ by replacing certain subexpressions with others. We show that each of these replacements makes the sentence "truer," in the sense that it produces a sentence implied by the original.

We consider separately three kinds of replacement:

- Replacing a positive subsentence of form $e\langle x, y\rangle \tilde{\prec} e'\langle x, y\rangle$ with $e\langle y, x\rangle \tilde{\prec} e'\langle y, x\rangle$, where the replaced occurrences of x and y satisfy the strengthening conditions in $e\langle x, y\rangle$ and $e'\langle x, y\rangle$ with respect to $\prec$ and $\tilde{\prec}$.

  In this case, because $x \prec y$, we have (by the *strengthened polarity-replacement* lemma) that

  $$\begin{array}{l} \textit{if } e\langle x, y\rangle \tilde{\prec} e'\langle x, y\rangle \\ \textit{then } e\langle y, x\rangle \tilde{\prec} e'\langle y, x\rangle. \end{array}$$

Therefore, because the replaced occurrence of $e\langle x, y\rangle \tilde{\prec} e'\langle x, y\rangle$ is positive in $\mathcal{P}\langle x^+, y^-\rangle$, we know (by the original *polarity-replacement* proposition) that replacing it with the "truer" subsentence $e\langle y, x\rangle \tilde{\prec} e'\langle y, x\rangle$ makes the entire sentence truer.

- Replacing a negative subsentence of form $e\langle x, y\rangle \tilde{\prec} e'\langle x, y\rangle$, with $e\langle y, x\rangle \tilde{\preceq} e'\langle y, x\rangle$, where the replaced occurrences of x and y satisfy the strengthening conditions in $e\langle x, y\rangle$ and $e'\langle x, y\rangle$ with respect to $\prec$ and $\tilde{\nprec}$ (the negation of $\tilde{\prec}$).

  In this case, because $x \prec y$, we have (by the *strengthened polarity-replacement* lemma, recalling that $\tilde{\npreceq}$ is the irreflexive restriction of $\tilde{\nprec}$)

  $$\begin{array}{l} \textit{if } e\langle x, y\rangle \tilde{\nprec} e'\langle x, y\rangle \\ \textit{then } e\langle y, x\rangle \tilde{\npreceq} e'\langle y, x\rangle \end{array}$$

or, equivalently (taking the contrapositive),

$$\begin{array}{l} \textit{if } e\langle y, x\rangle \tilde{\preceq} e'\langle y, x\rangle \\ \textit{then } e\langle x, y\rangle \tilde{\prec} e'\langle x, y\rangle. \end{array}$$

Therefore, because the replaced occurrence of $e\langle x, y\rangle \tilde{\prec} e'\langle x, y\rangle$ is negative in $\mathcal{P}\langle x^+, y^-\rangle$, we know (by the original *polarity-replacement* proposition) that replacing it with the "falser" sentence $e\langle y, x\rangle \tilde{\preceq} e'\langle y, x\rangle$ will make the entire sentence falser.

- Replacing a positive occurrence of x with y or a negative occurrence of y with x, where polarity is with respect to $\prec$ and where the replaced occurrence is not within the scope of any strengthened relation $\tilde{\prec}$.

  In this case, the replacement makes the sentence "truer," by the original *polarity-replacement* proposition. ┘

THE GROUND VERSION

We can now express the stronger version of the relation replacement rule. The ground version of the rule is as follows:

Rule (strengthened relation replacement, ground version)

For any binary relation $\prec$, ground expressions s and t, and ground sentences $\mathcal{F}[s \prec t]$ and $\mathcal{G}\langle s^+, t^-\rangle$, we have

$$\begin{array}{c} \mathcal{F}[s \prec t] \\ \mathcal{G}\langle s^+,\ t^-\rangle \\ \hline \mathcal{F}[\textit{false}] \ \textit{or} \ \mathcal{G}\langle t^+,\ s^-\rangle^{\dagger} \end{array}$$

Here $\mathcal{G}\langle t^{+}, s^{-}\rangle^{\dagger}$ is the result of replacing certain positive occurrences of s with t, replacing certain negative occurrences of t with s, and strengthening accordingly, where polarity is taken in $\mathcal{G}\langle s^{+}, t^{-}\rangle$ with respect to $\prec$. We assume that at least one replacement is made. ▟

Let us illustrate the ground version of the rule with two examples.

Example

In the theory of the positive integers (excluding 0), suppose our deduced set contains the sentences

$$\mathcal{F}: \quad \mathit{if}\ p(s)\ \mathit{then}\ \boxed{s} \prec_{div} t$$

and

$$\mathcal{G}: \quad a \le (\boxed{s}^{+} + 1)^2 \ \mathit{or}\ q(s),$$

where $\prec_{div}$ is the proper divides relation. Then we can apply the strengthened $\prec_{div}$-replacement rule to replace the boxed occurrence of s in $\mathcal{G}$ with t and to strengthen accordingly, obtaining

$$\begin{array}{l} \mathit{if}\ p(s)\ \mathit{then}\ \mathit{false} \\ \quad \mathit{or} \\ a < (t+1)^2 \ \mathit{or}\ q(s). \end{array}$$

This sentence reduces under transformation to

$$(\mathit{not}\ p(s))\ \mathit{or}\ a < (t+1)^2\ \mathit{or}\ q(s).$$

The relation symbol $\le$ was replaced by its irreflexive restriction $<$ because $a \le (s+1)^2$ is positive and because s and t satisfy the strengthening conditions in a and $(s+1)^2$ with respect to $\prec_{div}$ and $\le$, as we have seen in a previous example.

Example

In a theory that includes the sets and the nonnegative integers, suppose our deduced set contains the sentences

$$\mathcal{F}: \quad p(s,t)\ \mathit{or}\ \boxed{s} \subset \boxed{t}$$

and

$$\mathcal{G}: \quad \mathit{not}\left(q(s,t)\ \mathit{and}\ m\ \mathit{card}(\boxed{s}^{+}) < n + \mathit{card}(\boxed{t}^{-})\right),$$

where s and t are sets, m and n are nonnegative integers, and $\mathit{card}(s)$ is the cardinality of the set s. Then we can apply the strengthened $\subset$-replacement rule to replace the boxed occurrences of s with t and t with s and to strengthen accordingly, obtaining

$$\begin{array}{l} p(s,t)\ \mathit{or}\ \mathit{false} \\ \quad \mathit{or} \\ \mathit{not}\left(q(s,t)\ \mathit{and}\ m\ \mathit{card}(t) \le n + \mathit{card}(s)\right), \end{array}$$

that is (after transformation),

$$\begin{array}{l} p(s,t)\ \mathit{or} \\ \mathit{not}\left(q(s,t)\ \mathit{and}\ m \cdot \mathit{card}(t) \le n + \mathit{card}(s)\right). \end{array}$$

The relation symbol $<$ has been replaced by its reflexive closure $\leq$ because $m \cdot card(s) < n + card(t)$ is negative and because s and t satisfy the strengthening conditions in $m \cdot card(s)$ and $n + card(t)$ with respect to $\subset$ and $\not<$, that is, $\geq$. In particular,

- The irreflexive restriction $>$ of $\geq$ is transitive; hence the *transitivity* condition is satisfied.
- The replaced occurrences of s and t are respectively positive and negative in $m \cdot card(s) \leq n + card(t)$ with respect to $\subset$ and $\geq$; hence the *top* condition is satisfied.
- The replaced occurrence of t is negative in $n + card(t)$ with respect to $\subset$ and $>$; hence the *right* disjunct of the *left-right* condition is satisfied. ⌟

Let us now establish the soundness of the rule.

Justification (relation replacement rule, ground version)

The proof resembles the justification of the original relation-replacement rule.

We suppose that the given sentences $\mathcal{F}[s \prec t]$ and $\mathcal{G}\langle s^+,\ t^-\rangle$ are true and show that the newly deduced sentence $(\mathcal{F}[false]\ or\ \mathcal{G}\langle t^+,\ s^-\rangle^\dagger)$ is also true. We distinguish between two cases and show that in each case one of the two disjuncts, $\mathcal{F}[false]$ or $\mathcal{G}\langle t^+,\ s^-\rangle^\dagger$, is true.

In the case in which the subsentence $s \prec t$ is false, we know (by the value property, because $s \prec t$ and *false* have the same truth value and $\mathcal{F}[s \prec t]$ is true) that the first of the disjuncts, $\mathcal{F}[false]$, is true.

In the case in which $s \prec t$ is true, we know (by the ***strengthened polarity-replacement*** proposition, because $\mathcal{G}\langle s^+,\ t^-\rangle$ is true) that the second of the disjuncts, $\mathcal{G}\langle t^+,\ s^-\rangle^\dagger$, is true. ⌟

THE GENERAL VERSION

The general version of the rule allows us to instantiate the variables of the sentences as necessary to create common subexpressions.

Rule (strengthened relation replacement, general version)

For any binary relation $\prec$, expressions s, t, $\tilde{s}$, and $\tilde{t}$, and sentences $\mathcal{F}[s \prec t]$ and $\mathcal{G}\langle \tilde{s}^+,\ \tilde{t}^+\rangle$, where $\mathcal{F}$ and $\mathcal{G}$ are standardized apart, we have

$$\frac{\begin{array}{c}\mathcal{F}[s \prec t]\\ \mathcal{G}\langle \tilde{s}^+,\ \tilde{t}^-\rangle\end{array}}{\mathcal{F}\theta[false]\ \ or\ \ \mathcal{G}\theta\langle t\theta^+,\ s\theta^-\rangle^\dagger,}$$

where θ is a simultaneous, most-general unifier of s, $\tilde{s}$ and of t, $\tilde{t}$. ⌟

As usual, to apply the general version of the rule to sentences $\mathcal{F}$ and $\mathcal{G}$, we apply its ground version to $\mathcal{F}\theta$ and $\mathcal{G}\theta$. The justification, which is straightforward, is omitted. As before, the polarity strategy for the rule allows us to assume that a least one occurrence of the subsentence $(s \prec t)\theta$ is positive or of no polarity in $\mathcal{F}\theta$.

8. EXTENSIONS

The concepts in this paper are being extended in several directions. We briefly indicate several of these here.

EXPLICIT QUANTIFIERS

The system we have described deals with sentences that have had their quantifiers removed by skolemization. It is impossible, however, to remove quantifiers that occur within the scope of an equivalence ($\equiv$) connective or in the *if*-clause of a conditional (*if-then-else*) connective without first paraphrasing the connective in terms of others. If several of these connectives are nested, the paraphrased sentence becomes alarmingly complex.

In an earlier work (Manna and Waldinger [82]), we extend the deductive system to sentences that may have some of their quantifiers intact. In many cases, we can complete the proof without removing all the quantifiers. If these quantifiers are in equivalences or *if*-clauses, we need not paraphrase the offending connectives. Thus, we not only retain the form of the original sentence, but also can use the equivalences we retain in applying the equivalence replacement rule.

POLARITY WITH RESPECT TO AN EXPRESSION

We have used the notion of polarity with respect to a relation. Because a function is a special case of a relation, we can define polarity with respect to a function accordingly. Rather than restricting ourselves to the functions denoted by the function symbols in our deduced set, we prefer to consider the functions corresponding to particular expressions in the set.

Roughly speaking, suppose $e[s]$ is a ground term; then $e[s]$ corresponds to a binary relation $\prec_{e[s]}$ defined by the sentence

$$x \prec_{e[s]} y \;\equiv\; e[x] = y.$$

We may define polarity with respect to $\prec_{e[s]}$ just as we would with respect to any binary relation.

For example, in the theory of the integers, the relation $\prec_{e[s]}$ corresponding to the term $e[s] : s + 1$ is defined by the sentence

$$x \prec_{e[s]} y \;\equiv\; x + 1 = y.$$

(In fact, this relation turns out to be the predecessor relation $\prec_{pred}$ we have seen earlier.) The relation *natnum*(x), which holds if x is a nonnegative integer (natural number), is positive over its argument with respect to $\prec_{e[s]}$, for we have

```
if $x \prec_{e[s]} y$
then if $natnum(x)$
     then $natnum(y)$.
```

We can then establish an expression replacement rule analogous to our relation replacement rule; i.e., in the ground version:

For any expressions s and $e[s]$ and ground sentence $\mathcal{G}\langle s^+,\, e[s]^-\rangle$, we have

$$\frac{\mathcal{G}\langle s^+,\, e[s]^-\rangle}{\mathcal{G}\langle e[s]^+,\, s^-\rangle^{\dagger}}$$

Here $\mathcal{G}\langle e[s]^+,\ s^-\rangle^\dagger$ is obtained from $\mathcal{G}\langle s^+,\ e[s]^-\rangle$ by replacing certain positive occurrences of s with $e[s]$, replacing certain negative occurrences of $e[s]$ with s, and strengthening accordingly, where polarity is taken in $\mathcal{G}\langle s^+,\ e[s]^-\rangle$ with respect to $\prec_{e[s]}$.

For example, in the theory of the integers, if our deduced set contains the sentence

$$\mathcal{G}: \quad \mathit{not}\ \left[\mathit{natnum}\big((s+1)^-\big)\right]$$

we may deduce the sentence

$$\mathit{not}\ \left[\mathit{natnum}(s)\right],$$

because the occurrence of $s+1$ is negative in $\mathcal{G}$ with respect to the relation corresponding to the expression $s+1$.

We can also define *expression-matching rules analogous to our relation-matching rule.*

For example, in the theory of lists, suppose our deduced set contains the sentences

$$\mathcal{F}: \quad \boxed{a \in s^+}$$

and

$$\mathcal{G}: \quad \mathit{not}\ \left(\boxed{a \in (b \circ s)^+}\right).$$

Here the term $b \circ s$ is the result of inserting the element b before the first element of the list s. By the resolution rule with expression matching, whose precise statement we omit, we may deduce (after transformation), the contradiction *false,* because s is positive in the boxed sentence $a \in s$ with respect to the relation corresponding to $b \circ s$.

CONDITIONAL POLARITY

Sometimes it is convenient to extend the notion of polarity to depend on the truth of certain conditions. For example, in the theory of integers (including negative integers) with respect to the relation $\leq$, the occurrence of s in the sentence

$$a < b \cdot s$$

might be regarded as positive if b is nonnegative and negative if b is nonpositive. (If b is 0, the occurrence might have both polarities). We could then adapt the relation replacement and relation matching rules to use this conditional polarity, imposing the appropriate conditions on whatever conclusion they draw.

More precisely, we define the notion of conditional polarity so that if x and y are respectively positive and negative in $\mathcal{P}\langle x^+,\ y^-\rangle$ with respect to the binary relation $\prec$ *subject to the condition* $\mathcal{H}[x, y, \mathcal{Q}]$, then the sentence

$$\mathcal{H}\left[x,\ y,\ \begin{array}{l} \mathit{if}\ x \prec y \\ \mathit{then}\ \mathit{if}\ \mathcal{P}\langle x^+,\ y^-\rangle \\ \qquad\ \mathit{then}\ \mathcal{P}\langle y^+,\ x^-\rangle^\dagger \end{array}\right]$$

is valid. Here $\mathcal{Q}$ denotes an arbitrary sentence; the indicated polarities of the replaced occurrences of x and y are subject to the condition $\mathcal{H}[x, y, \mathcal{Q}]$.

For example, according to this notion of conditional polarity, in the theory of the integers, the occurrence of x in the sentence

$$a \leq b + x^2$$

is positive with respect to the relation $<$ subject to the condition

$$\mathcal{H}[x, y, \mathcal{Q}]: \quad \begin{array}{l} \textit{if } x \geq 0 \\ \textit{then } \mathcal{Q}. \end{array}$$

Consequently, we have that the sentence

$$\begin{array}{l} \textit{if } x \geq 0 \\ \textit{then } \textit{if } x < y \\ \qquad \textit{then } \textit{if } a \leq b + x^2 \\ \qquad\qquad \textit{then } a < b + y^2 \end{array}$$

is valid. The relation $\leq$ was replaced by $<$ as the result of strengthening.

In terms of this notion, we can introduce conditional versions of the relation replacement rule and relation-matching rules. In particular, we have the conditional relation-replacement rule, i.e., in the ground version:

> For any binary relation $\prec$, ground expressions s and t, and ground sentences $\mathcal{F}[s \prec t]$ and $\mathcal{G}\langle s^+, t^- \rangle$, we have
>
> $$\frac{\begin{array}{c} \mathcal{F}[s \prec t] \\ \mathcal{G}\langle s^+, t^- \rangle \end{array}}{\mathcal{H}[s, t, \textit{false}] \ \textit{or} \ \mathcal{F}[\textit{false}] \ \textit{or} \ \mathcal{G}\langle t^+, s^- \rangle^{\dagger}.}$$
>
> Here the indicated polarities of the replaced occurrences of s and t are subject to the condition $\mathcal{H}[s, t, \mathcal{Q}]$.
>
> For example, in the theory of the integers, suppose our deduced set contains the sentences

$$\mathcal{F}: \quad \begin{array}{l} \textit{if } r(s,t) \\ \textit{then } s < t \end{array}$$

and

$$\mathcal{G}: \quad a < b \cdot s.$$

Note that the occurrence of s in $\mathcal{G}$ is positive with respect to the relation $<$ subject to the condition

$$\begin{array}{l} \textit{if } b \geq 0 \\ \textit{then } \mathcal{Q}. \end{array}$$

Therefore, according to the conditional $<$-replacement rule, we may deduce

$$\begin{bmatrix} \textit{if } b \geq 0 \\ \textit{then } \textit{false} \end{bmatrix} \ \textit{or} \ \begin{bmatrix} \textit{if } r(s,t) \\ \textit{then } \textit{false} \end{bmatrix} \ \textit{or} \ a < b \cdot t,$$

which reduces under transformation to

$$\big(\textit{not}\,(b \geq 0)\big) \ \textit{or} \ \big(\textit{not}\,\big(r(s,t)\big)\big) \ \textit{or} \ a < b \cdot t.$$

The conditional relation-matching rules are analogous. Of course these rules can be extended to apply to conditional polarity with respect to an expression rather than a relation.

PLANNING AND THE FRAME PROBLEM

Theorem-proving techniques have often been applied to problems in automatic planning. One approach to this application has been the formulation of a *situational logic*, a theory in which states of the world are regarded as domain elements, denoted by terms. Typically, an action in a plan is represented as a function mapping states into other states. The effects of an action can be described by axioms.

For example, the primary effect of putting one block on top of another is expressed by an axiom such as

$$\begin{array}{l} \textit{if } clear(x, w) \textit{ and } clear(y, w) \\ \textit{then } on(x, y, puton(x, y, w)). \end{array}$$

In other words, if block x is put on block y in a state w, then x will indeed be on y in the resulting state $puton(x, y, w)$. The antecedent expresses the preconditions that x and y be clear before x can be put on y; in other words, no block can be on x or on y. (The conventional blocks-world hand can move only one block at a time.)

In a situational logic, a problem may be expressed as a theorem to be proved. For example, the problem of achieving the condition that block a is on block b and block b is on block c might be phrased as the theorem

$$(\exists z)[on(a, b, z) \textit{ and } on(b, c, z)].$$

The *frame problem*, which occurs when planning problems are approached in this way, is connected with the requirement that we need to express not only what conditions are altered by a given action, but also what conditions are unchanged. For example, in addition to the primary effect of putting one block on top of another, we must state explicitly that this action has no effect on other relations, such as color; otherwise, we shall have no way of deducing that the color of a block after the action is the same as its color before. Therefore, we must include in our deduced set the *frame* axiom

$$\begin{array}{l} \textit{if } clear(x, w) \textit{ and } clear(y, w) \\ \textit{then if } color(z, u, w) \\ \qquad \textit{then } color(z, u, puton(x, y, w)). \end{array}$$

In other words, if the action of putting block x on top of block y is legal and if block z is of color u in state w, then z will also be of color u in the resulting state $puton(x, y, w)$. If our deduced set contains the sentence

$$not\,(color(c, red, puton(a, b, s))),$$

we can then apply the resolution rule to the frame axiom and this sentence to deduce (after transformation)

$$(not\,(clear(a, s))) \textit{ or } (not\,(clear(b, s))) \textit{ or } (not\,(color(c, red, s))).$$

We need a separate frame axiom not only for the color of blocks, but also their size, shape, surface texture, and any other attributes we wish to discuss in our theory. Adding all the frame axioms to our deduced set aggravates the search problem, because the axioms have many consequences irrelevant to the problem at hand.

By use of the conditional expression rules, we can drop all the frame axioms from our deduced set. For example, to paraphrase the above axiom we can declare that the relation $color(z, u, w)$ is positive with respect to the relation corresponding to the expression $e[w] : puton(x, y, w)$ subject to the condition

$$\mathcal{H}[w, w', \mathcal{Q}] : \quad \begin{array}{l} \textit{if } clear(x, w) \textit{ and } clear(y, w) \\ \textit{then } \mathcal{Q}. \end{array}$$

If our deduced set again contains the sentence

$$not\,(color(c, red, puton(a, b, s)^{-})),$$

we can then apply the conditional expression-replacement rule to deduce

$$(not\,(clear(a, s))) \textit{ or } (not\,(clear(b, s))) \textit{ or } (not\,(color(c, red, s)))$$

as before, without requiring the frame axiom. Of course, the information that certain actions and relations are independent must still be expressed, but this can be done by polarity declarations rather than by axioms.

9. DISCUSSION

The theorem-proving system we have presented has been motivated by our work in program synthesis, and the best examples we have of its use are in this domain. We have used the system to write detailed derivations for programs over the integers and real numbers, the lists, the sets, and other structures. These derivations are concise and easy to follow: they reflect intuitive derivations of the same programs. A paper by Traugott [85] describes the application of this system to the derivation of several sorting programs. A paper by Manna and Waldinger [85] describes the derivation of several binary-search programs. Our earlier informal derivation of the unification algorithm (Manna and Waldinger [81]) can be expressed formally in this system.

An interactive implementation of the basic nonclausal theorem-proving system was completed by Malachi and has been extended by Bronstein to include some of the relation rules. An entirely automatic implementation is being contemplated. The relation rules will also be valuable for proving purely mathematical theorems. For this purpose they may be incorporated into clausal as well as nonclausal theorem-proving systems.

Theorem provers have exhibited superhuman abilities in limited subject domains, but seem least competent in areas in which human intuition is best developed. One reason for this is that an axiomatic formalization obscures the simplicity of the subject area; facts that a person would consider too obvious to require saying in an intuitive argument must be stated explicitly and dealt with in the corresponding formal proof. A person who is easily able to conduct the argument informally may well be unable to understand the formal proof, let alone to produce it.

Our work in special relations is part of a continuing effort to make formal theorem proving resemble intuitive reasoning. In the kind of system we envision, proofs are shorter, the search space is compressed, and heuristics based on human intuition become applicable.

ACKNOWLEDGEMENTS

The authors would like to thank Martin Abadi, Alex Bronstein, Tomas Feder, Eric Muller, Neil Murray, David Plaisted, Mark Stickel, Jon Traugott, and Frank Yellin for their suggestions and careful reading. Jon Traugott suggested extending the notion of polarity from one relation to two, making the rules more powerful and the exposition simpler; he also proposed the extended notions of polarity with respect to an expression and conditional polarity. The manuscript was prepared by Evelyn Eldridge-Diaz with the TEX typesetting system.

REFERENCES

Anderson [70]
R. Anderson, Completeness results for E-resolution, *AFIPS Spring Joint Computer Conference*, 1970, pp. 652–656.

Boyer and Moore [79]
R. S. Boyer and J S. Moore, *A Computational Logic*, Academic Press, New York, N.Y., 1979.

Brand [75]
D. Brand, Proving theorems with the modification method, *SIAM Journal of Computing*, Vol. 4, No. 2, 1975, pp. 412–430.

Chang and Lee [73]

C. L. Chang and R. C. Lee, *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, New York, N.Y., 1973.

Digricoli [83]
V. Digricoli, *Resolution By Unification and Equality*, Ph.D. thesis, New York University, New York, N.Y., 1983.

Kowalski [79]
R. Kowalski, *Logic for Problem Solving*, North Holland, New York, N.Y., 1979.

Loveland [78]
D. W. Loveland, *Automated Theorem Proving: A Logical Basis*, North-Holland, New York, N.Y., 1978.

Manna and Waldinger [80]
Z. Manna and R. Waldinger, A deductive approach to program synthesis, *ACM Transactions on Programming Languages and Systems*, Vol. 2, No. 1, January 1980, pp. 90–121.

Manna and Waldinger [81]
Z. Manna and R. Waldinger, Deductive synthesis of the unification algorithm, *Science of Computer Programming*, Vol. 1, 1981, pp. 5–48.

Manna and Waldinger [82]
Z. Manna and R. Waldinger, Special relations in program-synthetic deduction, Technical Report, Computer Science Department, Stanford University, Stanford, Calif., and Artificial Intelligence Center, SRI International, Menlo Park, Calif., March 1982.

Manna, Z., and R. Waldinger [85a]

The Logical Basis for Computer Programming, Addison-Wesley, Reading, Mass., Volume 1: Deductive Reasoning (1985), Volume 2: Deductive Techniques (to appear).

Manna, Z., and R. Waldinger [85b]

The origin of the binary-search paradigm, *Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, August 1985.

Morris [69]
J. B. Morris, E-resolution: extension of resolution to include the equality relation, *International Joint Conference on Artificial Intelligence*, Washington, D.C., May 1969, pp. 287–294.

Murray [82]
N. V. Murray, Completely nonclausal theorem proving, *Artificial Intelligence*, Vol. 18, No. 1, 1982, pp. 67–85.

Robinson [65]
J. A. Robinson, A machine-oriented logic based on the resolution principle, *Journal of the ACM*, Vol. 12, No. 1, January 1965, pp. 23–41.

Robinson [79]
J. A. Robinson, *Logic: Form and Function*, North-Holland, New York, N.Y., 1979.

Stickel [82]
M. E. Stickel, A nonclausal connection-graph resolution theorem-proving program. *National Conference on AI*, Pittsburgh, Pa., 1982, pp. 229–233.

Traugott [85]
J. Traugott, Deductive synthesis of sorting algorithms, Technical Report, Computer Science Department, Stanford University, Stanford, Calif. (forthcoming).

Wos and Robinson [69]

L. Wos and G. Robinson, Paramodulation and theorem proving in first order theories with equality, in *Machine Intelligence 4* (B. Meltzer and D. Michie, editors) American Elsevier, New York, N.Y., 1969, pp. 135–150.